SANDIA REPORT

SAND2015-2428
Unlimited Release
Printed April 2, 2015

Preliminary Assessment of Tecplot Chorus for Analyzing Ensemble of CTH Simulations

Anthony M. Agelastos, Joel O. Stevenson, Stephen W. Attaway, David J. Peterson

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories

SAND2015-2428
Unlimited Release
Printed April 2, 2015

Preliminary Assessment of Tecplot Chorus for Analyzing Ensemble of CTH Simulations

Anthony M. Agelastos
amagela@sandia.gov

Joel O. Stevenson
josteve@sandia.gov

Stephen W. Attaway
swattaw@sandia.gov

David J. Peterson
djpeter@sandia.gov

Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185

Abstract

The exploration of large parameter spaces in search of problem solution and uncertainty quantification produces very large ensembles of data. Processing ensemble data will continue to require more resources as simulation complexity and HPC platform throughput increase. More tools are needed to help provide rapid insight into these data sets to decrease manual processing time by the analyst and to increase knowledge the data can provide. One such tool is Tecplot Chorus, whose strengths are visualizing ensemble metadata and linked images. This report contains the analysis and conclusions from evaluating Tecplot Chorus with an example problem that is relevant to Sandia National Laboratories.

Acknowledgment

The authors of this document acknowledge the contributions from the following individuals: Steven N. Kempka for providing funding for this investigation, John P. Korbin for assisting with the example ensemble data, and Ryan P. Shaw for informing this document's authors about zealous cropping. Also, Tecplot, Inc. was extremely responsive and helpful with their support of this evaluation.

Contents

List of Figures

List of Tables

Nomenclature

DOE	U.S. Department of Energy
SNL	Sandia National Laboratories
CSV	Comma-separated values (file that stores comma-delimited tabular data in plain text)
JPEG	Joint Photographic Experts Group (image file format with compression)
PNG	Portable Network Graphics (image file format with compression)
KB	Kilobyte, 1024^1 bytes
MB	Megabyte, 1024^2 bytes
GB	Gigabyte, 1024^3 bytes
HPC	High-performance computing
CEE	SNL's Common Engineering Environment (`https://prod.sandia.gov/cee/`)
LAN	Local area network

Chapter 1

Introduction

This report documents a preliminary evaluation of Tecplot Chorus for analyzing ensemble data from CTH simulations. The project that funded this report and evaluation is also evaluating and guiding development with SNL's Slycat[1]. Slycat and Tecplot Chorus each have their strengths, weaknesses, and overlapping capabilities. It is quite likely that, as the scale of ensemble data increases, both of these tools (and possibly others) will be needed for different processing goals. This report will focus on Tecplot Chorus and its application to an example ensemble of data supplied by David J. Peterson and John P. Korbin; this example is of a flyer plate impact and weld study henceforth referred to as *CTH Impact Example.*

This evaluation also defines a workflow for analysts that can help reduce the time and resources for processing ensemble data. This workflow's stages are enumerated below.

1. Minimize the footprint of ensemble metadata and supporting files, e.g., images. This stage will increase the portability of the ensemble in addition to increasing the efficiency that other tools, e.g., Tecplot Chorus, Slycat, can process it.

2. Create a database for each seed within the ensemble. This stage allows for seed-specific processing and for easy seed-based combining into larger databases.

3. Create a database for the entire ensemble. This stage creates a monolithic ensemble database that can either be processed with other tools or used as a single point to further filter.

4. Reduce the entire ensemble database using problem-specific filters. This stage will reduce the monolithic ensemble database into something that is manageable by the tools used for processing and for the analysts to comprehend.

This *Ensemble Workflow* will be referenced throughout this report. Defining a workflow allows for many focused tools to interact with one another for a common goal. Since ensemble processing will likely differ across projects, there will likely not be a one-size-fits-all tool or paradigm, but instead a collection of tools that are hopefully easy to integrate with each other to achieve the desired processing.

The list below describes each chapter and its intended audience.

Chapter 2 This chapter provides a SNL-centric overview of Tecplot Chorus. This chapter should be referenced by analysts who wish to use Chorus, system administrators who wish to install Chorus, and program managers who decide whether or not to procure Chorus.

Section 2.1 This section provides a general overview of Tecplot Chorus.

Section 2.2 This section provides a platform-centric analysis of Tecplot Chorus, including a description of the resources Chorus requires.

Section 2.3 This section provides pricing and support information for Tecplot Chorus.

Chapter 3 This chapter provides an overview of the *CTH Impact Example*. This chapter's target audience is for analysts, however it may be beneficial for others who would like to better understand how Chorus applies to a relevant workflow.

Section 3.1 This section provides a description of the *CTH Impact Example* ensemble.

Section 3.5 This section provides the step-by-step instructions for using Tecplot Chorus to analyze the *CTH Impact Example*.

Sections 3.2, 3.3, 3.4, 3.6 These sections provide background information for applying the *Ensemble Workflow* to the *CTH Impact Example*.

Chapter 4 This chapter provides the conclusions from this report along with future work for this project and related projects. This chapter's target audience is for analysts and program managers.

Appendix A This appendix provides guidance for batch processing image files to conserve resources, i.e., *Ensemble Workflow* 1.

Appendix B This appendix provides guidance for creating CSV files for each seed of the *CTH Impact Example*, i.e., *Ensemble Workflow* 2.

Appendix C This appendix provides guidance for creating monolithic CSV files, i.e., *Ensemble Workflow* 3.

Appendix D This appendix provides guidance for filtering CSV files, i.e., *Ensemble Workflow* 4.

Appendix E This appendix provides additional background information on `csvcreate.py`, a Python script that was created to interface Tecplot Chorus with the *CTH Impact Example*.

Appendix F This appendix provides the example codes given from Tecplot to show how to extend Chorus with user functions.

Appendix G This appendix provides guidance for obtaining the source code for this report and the scripts contained within.

Chapter 2

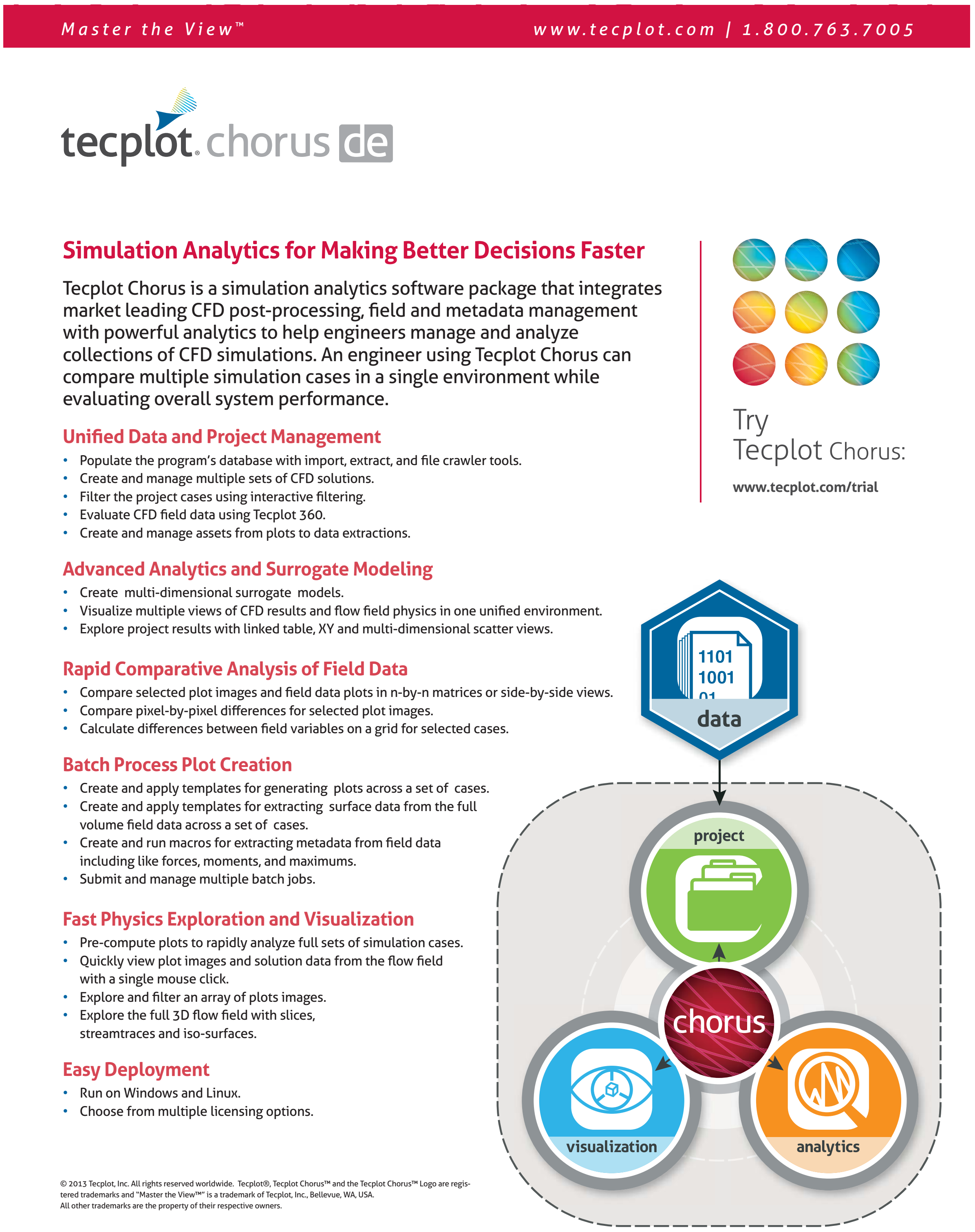
Tecplot Chorus

This chapter provides a brief introduction to Tecplot Chorus.

2.1 Description

Tecplot, Inc.[2] develops visualization products. Their product for visualizing ensemble data is Tecplot Chorus[3]; the version of Chorus at the time of this writing is DE 2013 Release 1. Chorus is meant to analyze simulation metadata, incorporate simulation images, and interface with their other products, e.g., Tecplot 360[4], to provide the end user with broad tools for analyzing ensemble data. A comprehensive description of Tecplot Chorus is given in the data sheet[5] supplied from Tecplot, Inc. shown in Figure 2.1; page 1 of the data sheet is Figure 2.1a and page 2 of the data sheet is Figure 2.1b.

One feature not mentioned within the data sheets is that Chorus' functionality can be extended through the use of Python scripts. Tecplot provided some examples of extended functionality; these code snippets are provided within Appendix F for reference. An example scenario that is relevant for CTH ensemble processing is to be able to right-click on a point and to tell it to generate an animation from the available Spymaster output within that directory with CTH's `makempeg` function.



Master the View™ *www.tecplot.com | 1.800.763.7005*

tecplot. chorus de

Simulation Analytics for Making Better Decisions Faster

Tecplot Chorus is a simulation analytics software package that integrates market leading CFD post-processing, field and metadata management with powerful analytics to help engineers manage and analyze collections of CFD simulations. An engineer using Tecplot Chorus can compare multiple simulation cases in a single environment while evaluating overall system performance.

Unified Data and Project Management

- Populate the program's database with import, extract, and file crawler tools.
- Create and manage multiple sets of CFD solutions.
- Filter the project cases using interactive filtering.
- Evaluate CFD field data using Tecplot 360.
- Create and manage assets from plots to data extractions.

Advanced Analytics and Surrogate Modeling

- Create multi-dimensional surrogate models.
- Visualize multiple views of CFD results and flow field physics in one unified environment.
- Explore project results with linked table, XY and multi-dimensional scatter views.

Rapid Comparative Analysis of Field Data

- Compare selected plot images and field data plots in n-by-n matrices or side-by-side views.
- Compare pixel-by-pixel differences for selected plot images.
- Calculate differences between field variables on a grid for selected cases.

Batch Process Plot Creation

- Create and apply templates for generating plots across a set of cases.
- Create and apply templates for extracting surface data from the full volume field data across a set of cases.
- Create and run macros for extracting metadata from field data including like forces, moments, and maximums.
- Submit and manage multiple batch jobs.

Fast Physics Exploration and Visualization

- Pre-compute plots to rapidly analyze full sets of simulation cases.
- Quickly view plot images and solution data from the flow field with a single mouse click.
- Explore and filter an array of plots images.
- Explore the full 3D flow field with slices, streamtraces and iso-surfaces.

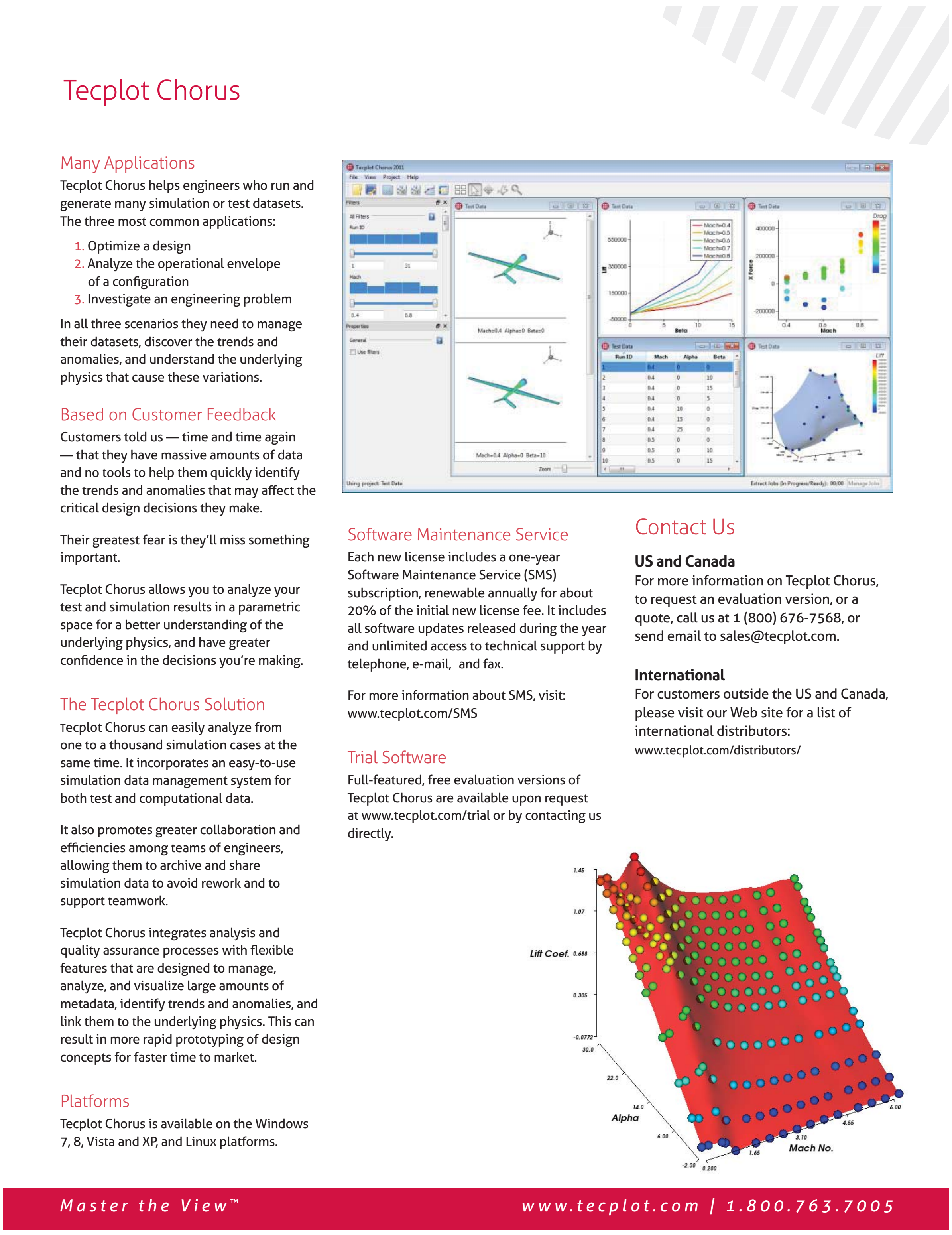
Easy Deployment

- Run on Windows and Linux.
- Choose from multiple licensing options.

Try
Tecplot Chorus:

www.tecplot.com/trial

(a) Page 1



Tecplot Chorus

Many Applications

Tecplot Chorus helps engineers who run and generate many simulation or test datasets. The three most common applications:

1. Optimize a design
2. Analyze the operational envelope of a configuration
3. Investigate an engineering problem

In all three scenarios they need to manage their datasets, discover the trends and anomalies, and understand the underlying physics that cause these variations.

Based on Customer Feedback

Customers told us — time and time again — that they have massive amounts of data and no tools to help them quickly identify the trends and anomalies that may affect the critical design decisions they make.

Their greatest fear is they'll miss something important.

Tecplot Chorus allows you to analyze your test and simulation results in a parametric space for a better understanding of the underlying physics, and have greater confidence in the decisions you're making.

The Tecplot Chorus Solution

Tecplot Chorus can easily analyze from one to a thousand simulation cases at the same time. It incorporates an easy-to-use simulation data management system for both test and computational data.

It also promotes greater collaboration and efficiencies among teams of engineers, allowing them to archive and share simulation data to avoid rework and to support teamwork.

Tecplot Chorus integrates analysis and quality assurance processes with flexible features that are designed to manage, analyze, and visualize large amounts of metadata, identify trends and anomalies, and link them to the underlying physics. This can result in more rapid prototyping of design concepts for faster time to market.

Platforms

Tecplot Chorus is available on the Windows 7, 8, Vista and XP, and Linux platforms.

Software Maintenance Service

Each new license includes a one-year Software Maintenance Service (SMS) subscription, renewable annually for about 20% of the initial new license fee. It includes all software updates released during the year and unlimited access to technical support by telephone, e-mail, and fax.

For more information about SMS, visit: www.tecplot.com/SMS

Trial Software

Full-featured, free evaluation versions of Tecplot Chorus are available upon request at www.tecplot.com/trial or by contacting us directly.

Contact Us

US and Canada
For more information on Tecplot Chorus, to request an evaluation version, or a quote, call us at 1 (800) 676-7568, or send email to sales@tecplot.com.

International
For customers outside the US and Canada, please visit our Web site for a list of international distributors: www.tecplot.com/distributors/

Run ID	Mach	Alpha	Beta
1	0.4	0	0
2	0.4	0	10
3	0.4	0	15
4	0.4	0	5
5	0.4	10	0
6	0.4	15	0
7	0.4	25	0
8	0.5	0	0
9	0.5	0	10
10	0.5	0	15

Master the View™ www.tecplot.com | 1.800.763.7005

(b) Page 2

Figure 2.1: Tecplot Chorus DE 2013 Release 1 vendor data sheet

2.2 System and Resource Impact

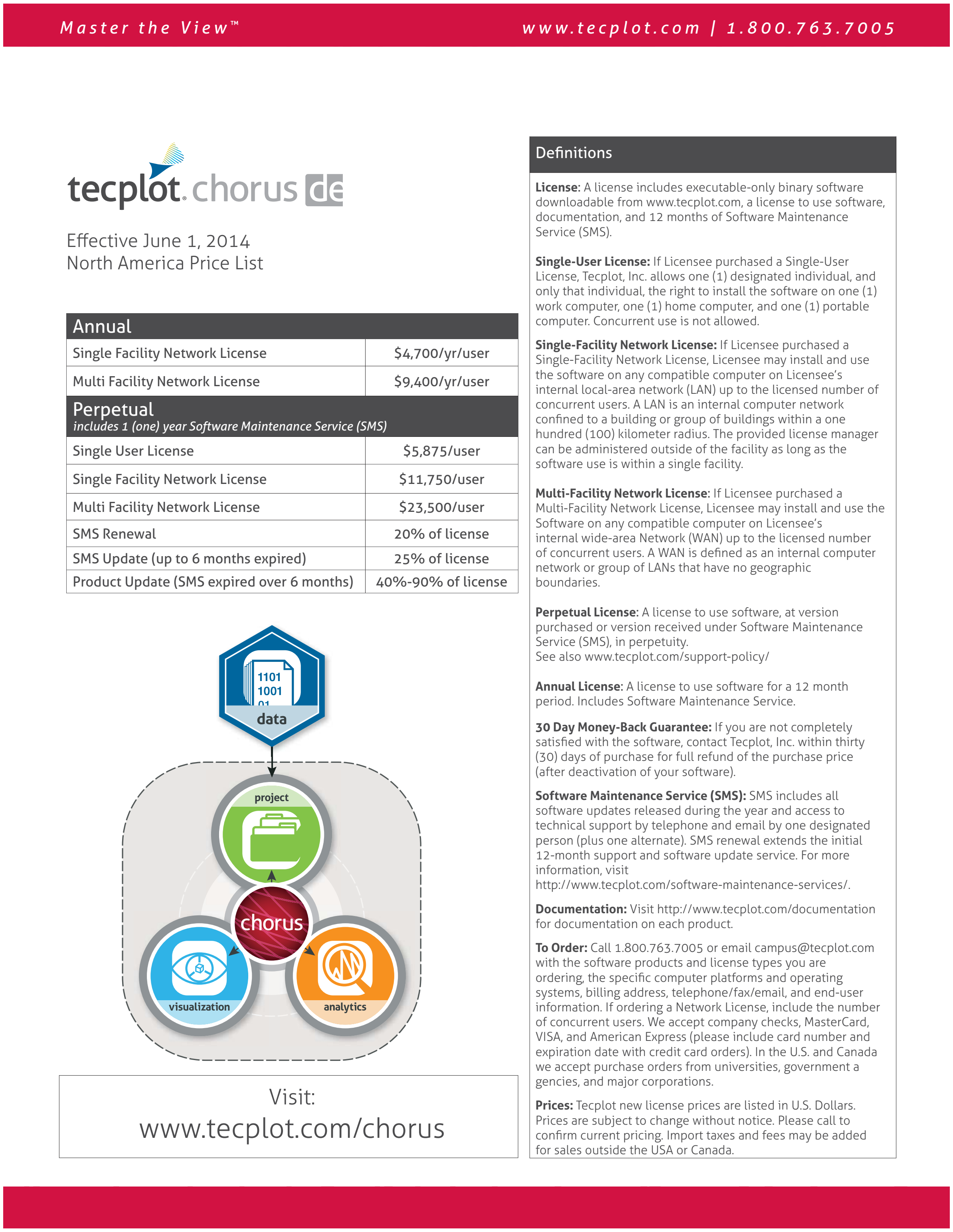
The following list contains some system- and resource-centric observations from installing and using Tecplot Chorus on the *CTH Impact Example* in Chapter 3.

1. Chorus can only see data on the system it is running on.
2. To use Chorus on a local computer, it would have to be running on the local computer or it would have to run on a different system and be displayed on the local computer by X11 forwarding or screen capture technology, e.g., VNC, NX, RGS. SNL's LAN is sufficient for X11 forwarding as long as both systems are within SNL.
3. Chorus will load all of the ensemble metadata provided to it into memory and will only load images upon request. Chorus' RAM requirements will grow as the metadata sizes grow. The *CTH Impact Example* monolithic CSV file is approximately 753 MB and contains 1,191 columns and 53,424 rows of data, totaling 63,627,984 total data points. When Chorus loads this into memory, it is using upwards of 26 GB of RAM, which is more than what is available on most laptop and desktop systems.
4. Chorus is unable to properly save its own Project and Session files on the monolithic CSV file since it wants to save out a file that is larger than 2 GB and it was unable to; 32-bit issues within the code are the likely culprit.
5. After installation on a Linux system, an analyst only needs to add the installation binary path to their own `PATH` environment variable. As a result, installing Chorus on HPC platforms and interfacing with it via modules would be very easy.

2.3 Support and Pricing

The following list provides some support and pricing data for Tecplot Chorus.

- During this evaluation, Tecplot, Inc. has made its members available for video conferences, telephone calls, and responding to many emails, all to either answer questions or provide their own guided tutorials on Chorus features.
- Tecplot's technical support has been very timely, e.g., during this evaluation a CSV file was loaded into Chorus causing awkward behavior. The root cause of this was the CSV file had duplicate-named columns. Within 24 hours of reporting the awkward behavior, Tecplot had a patched version of Chorus available for download.
- Tecplot would like a SNL member of the workforce to be on their Council to help guide development roadmaps and feature priority for their tools.
- The current pricing for Tecplot Chorus is shown in Figure 2.2.

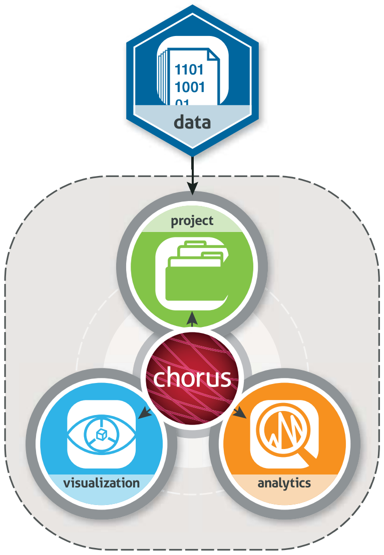


Master the View™ www.tecplot.com | 1.800.763.7005

tecplot chorus de

Effective June 1, 2014
North America Price List

Annual	
Single Facility Network License	$4,700/yr/user
Multi Facility Network License	$9,400/yr/user
Perpetual *includes 1 (one) year Software Maintenance Service (SMS)*	
Single User License	$5,875/user
Single Facility Network License	$11,750/user
Multi Facility Network License	$23,500/user
SMS Renewal	20% of license
SMS Update (up to 6 months expired)	25% of license
Product Update (SMS expired over 6 months)	40%-90% of license



Visit:
www.tecplot.com/chorus

Definitions

License: A license includes executable-only binary software downloadable from www.tecplot.com, a license to use software, documentation, and 12 months of Software Maintenance Service (SMS).

Single-User License: If Licensee purchased a Single-User License, Tecplot, Inc. allows one (1) designated individual, and only that individual, the right to install the software on one (1) work computer, one (1) home computer, and one (1) portable computer. Concurrent use is not allowed.

Single-Facility Network License: If Licensee purchased a Single-Facility Network License, Licensee may install and use the software on any compatible computer on Licensee's internal local-area network (LAN) up to the licensed number of concurrent users. A LAN is an internal computer network confined to a building or group of buildings within a one hundred (100) kilometer radius. The provided license manager can be administered outside of the facility as long as the software use is within a single facility.

Multi-Facility Network License: If Licensee purchased a Multi-Facility Network License, Licensee may install and use the Software on any compatible computer on Licensee's internal wide-area Network (WAN) up to the licensed number of concurrent users. A WAN is defined as an internal computer network or group of LANs that have no geographic boundaries.

Perpetual License: A license to use software, at version purchased or version received under Software Maintenance Service (SMS), in perpetuity.
See also www.tecplot.com/support-policy/

Annual License: A license to use software for a 12 month period. Includes Software Maintenance Service.

30 Day Money-Back Guarantee: If you are not completely satisfied with the software, contact Tecplot, Inc. within thirty (30) days of purchase for full refund of the purchase price (after deactivation of your software).

Software Maintenance Service (SMS): SMS includes all software updates released during the year and access to technical support by telephone and email by one designated person (plus one alternate). SMS renewal extends the initial 12-month support and software update service. For more information, visit http://www.tecplot.com/software-maintenance-services/.

Documentation: Visit http://www.tecplot.com/documentation for documentation on each product.

To Order: Call 1.800.763.7005 or email campus@tecplot.com with the software products and license types you are ordering, the specific computer platforms and operating systems, billing address, telephone/fax/email, and end-user information. If ordering a Network License, include the number of concurrent users. We accept company checks, MasterCard, VISA, and American Express (please include card number and expiration date with credit card orders). In the U.S. and Canada we accept purchase orders from universities, government a gencies, and major corporations.

Prices: Tecplot new license prices are listed in U.S. Dollars. Prices are subject to change without notice. Please call to confirm current pricing. Import taxes and fees may be added for sales outside the USA or Canada.

Figure 2.2: North America price list for Tecplot Chorus as of June 1st, 2014

Chapter 3

CTH Impact Example

This chapter describes the *CTH Impact Example* and how it was post processed by Tecplot Chorus.

3.1 Description

The *CTH Impact Example* utilizes SNL's Dakota[6] to drive a parametric set of SNL CTH[7] simulations. The primary purpose of this study is to determine explosive welding configurations that best match known regions of weld phenomena identified from previous test data. To this end, the study simulates a flat, semi-infinite witness plate with a flyer plate of varying thickness at a set distance above it mating to a varying thickness of Detasheet 2000. This initial configuration is shown in Figure 3.1 below. Along with these thickness parameters a mechanical angle can also be varied. The net result is that a flyer of varying thickness can be thrown at varying velocities and angles. The configuration that is shown to have the earliest arrival time within the correct explosive+mechanical attack angle window for welding (i.e., 5-14 deg at 1-2 km/s) is an optimal design choice. Notably this parametric study should also be capable of ruling out spallation conditions that would be unsuitable for use. The initial version of this ensemble has 100 Dakota evaluations, or seeds. To help determine earliest arrival times and visually inspect the ensemble for viable, nonspalled configurations, David J. Peterson, an author of this ensemble, wishes to achieve the following goals with the aid of Tecplot Chorus.

1. Create a CSV file that contains all data from the CTH `hscth` file, the Dakota `params.in` file, and the Spymaster images.
2. Visually inspect images to find, or verify, time of impact.
3. Determine the impact location.
4. Compute relative angle of impact.

The *Ensemble Workflow* described within Chapter 1 will be applied to these goals in the following sections. A Python program named `csvcreate.py` was created to help facilitate loading *CTH Impact Example* into Tecplot Chorus.

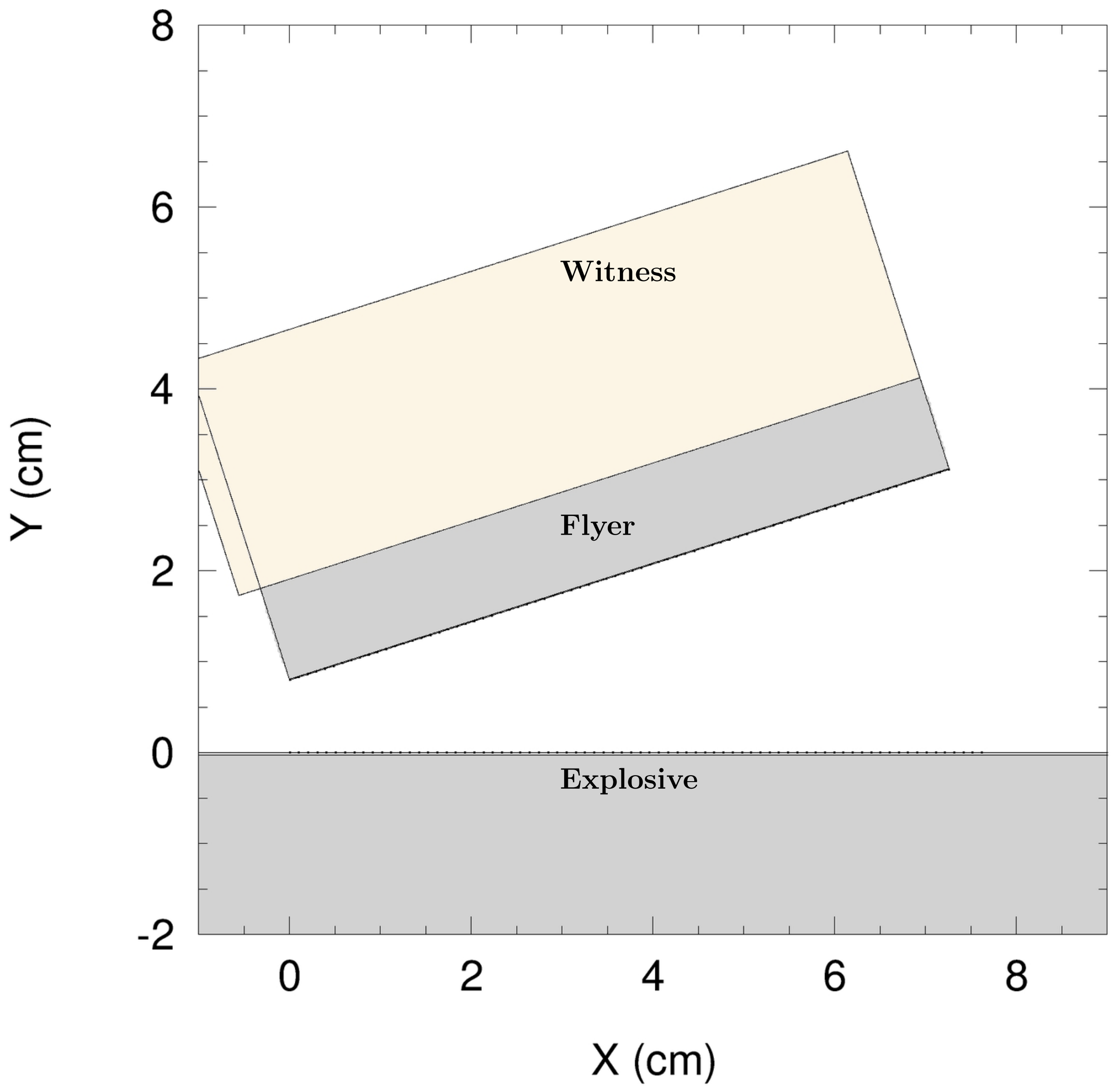


Figure 3.1: Initial configuration of explosively propelled flyer above witness plate

3.2 Workflow Stage 1: Footprint Reduction

The 1st stage of the *Ensemble Workflow*, which was previously enumerated within Chapter 1, is to minimize the footprint of ensemble metadata and supporting files, e.g., images. This stage will increase the portability of the ensemble in addition to increasing the efficiency with which other tools, e.g., Tecplot Chorus, Slycat, can process the ensemble. This stage will create new images that are space efficient. Please refer to Appendix A for detailed instructions and background information for performing this stage. While this stage is achieved with the command `` `csvcreate.py -O` `` and requires almost no analyst effort, there is still time spent waiting for the conversions (a compressed copy is made for each image file) to occur. This could be minimized as well if the conversion was built into the simulation workflow. A typical wall time for this step with *CTH Impact Example*, from scratch, is around 4 hours with full access to a system with 64 cores. Afterwards, there are 534,240 new `workdir.*/*.png` files.

3.3 Workflow Stage 2: Create CSV File for Each Seed

The 2nd stage of the *Ensemble Workflow* is to create a database for each seed within the ensemble. This stage allows for seed-specific processing and for easy seed-based combining into larger databases. This stage will create one CSV file per seed that correlates the `hscth`, `params.in`, and Spymaster image files. Please refer to Appendix B for detailed instructions and background information for performing this stage. Ultimately, this stage is achieved with the command `` `csvcreate.py -1 -f PNG -n 2` `` and requires almost no analyst effort. The time spent waiting for this step to complete is dependent upon the scale of the file system it must crawl and how responsive the underlying file system is; typical wall times for this step with *CTH Impact Example* are around 30 minutes. Afterwards, there are new `workdir.*/hscth.individual.csv` files.

3.4 Workflow Stage 3: Create Combined CSV Files

The 3rd stage of the *Ensemble Workflow* is to create a database for the entire ensemble. This stage creates a monolithic ensemble database that can either be processed with other tools or used as a single point to further filter. this stage will create a monolithic CSV file, named `hscth.combined.csv` and several "piecemeal" files, named `hscth.combined#.csv`, that are pieces of the monolithic one. Please refer to Appendix C for detailed instructions and background information for performing this stage. Ultimately, this stage is achieved with the command `` `csvcreate.py -2` `` and requires almost no analyst effort. This command is very quick and requires approximately 1 minute to complete.

3.5 Processing Available Data with Tecplot Chorus

After performing the steps within Sections 3.2, 3.3, and 3.4, there are several types of CSV files available for processing. These types are listed below.

`workdir.*/hscth.individual.csv`:	There is one of these CSV files for each seed.
`hscth.combined*.csv`:	There are 10 of these files where the 1st one contains data from `workdir.1` through `workdir.10`, the 2nd one contains data from `workdir.11` through `workdir.20`, and so forth.

`hscth.combined.csv`: This file contains the individual data from all `workdir` directories.

Since all of these files are present, goal 1 is complete and goal 2 can be focused upon. After executing the commands discussed in the previous sections, each of these CSV files contain the following columnar data not initially present within the `hscth` file. This data and their descriptions are listed below.

`Identifier`: This is the directory name that the `hscth` file was found within. If this name is a typical Dakota working directory name, e.g., `workdir.1`, then it will only contain the seed integer number rather than the whole name, e.g., `1`.

`OutputNum`: This is the line number from that row's `hscth.individual.csv` file where it can be found.

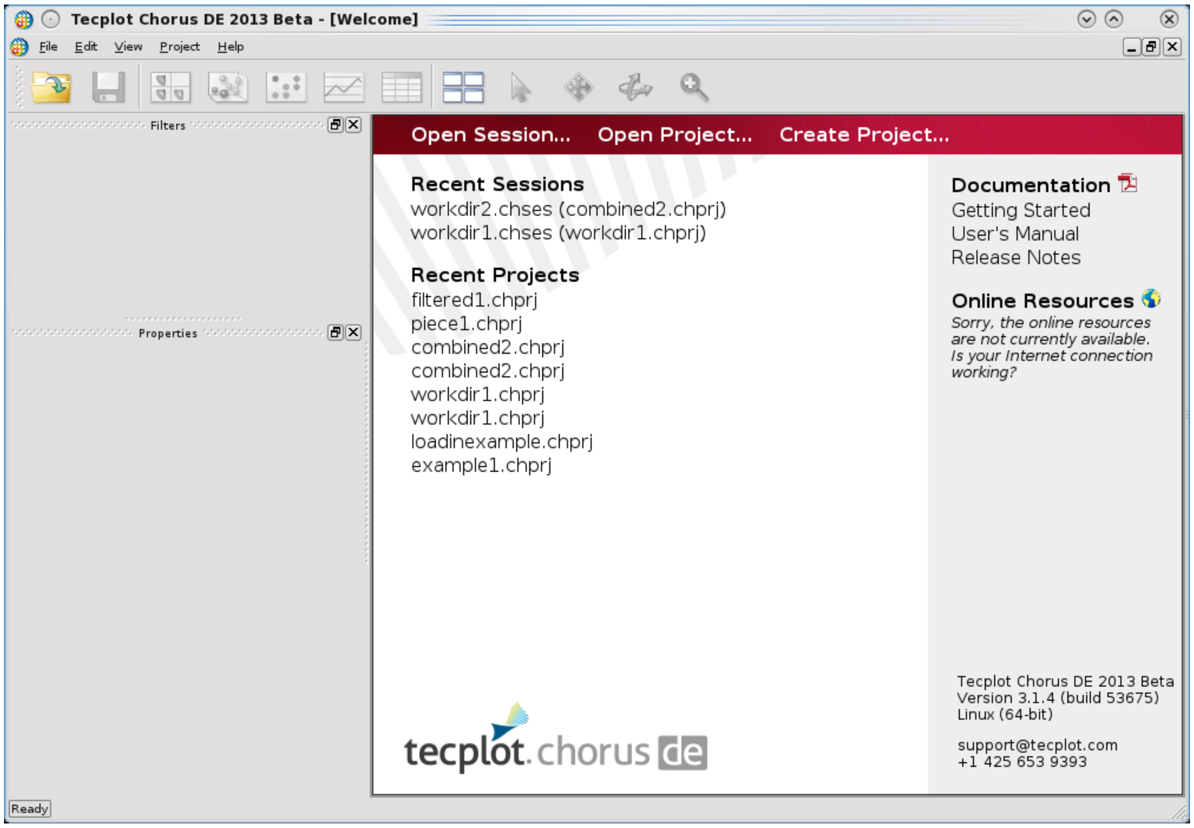
`angle_of_awesome, exp_thick, fly_thick`: These are independent variables pulled in from the Dakota `params.in` file; these variables correspond to the reference angle, explosive thickness, and flyer thickness, respectively.

`User1, User0`: These are the 2 user variables created with the "`-n 2`" argument passed in Section 3.3.

`shap-Damage...blocks-shap-Position`: These are the 10 different families of Spymaster output for *CTH Impact Example*; in this case the PNG compressed output is referenced, created from the argument "`-f PNG`" passed in Section 3.2, instead of the default JPEG Spymaster output.

Figures 3.2-3.10 and their captions provide a step-by-step tutorial of importing any one of the CSV files generated into Tecplot Chorus and using it for goal 2, which is to determine the time of impact. These steps can be taken 100 times (1 for each `workdir.*/hscth.individual.csv` file), 10 times (1 for each `hscth.combined*.csv` file), or 1 time for the single `hscth.combined.csv` file. The load times increase as the CSV file size increases, so the optimum may change with different ensemble data. For *CTH Impact Example*, the "piecemeal" files, i.e., `hscth.combined*.csv`, were utilized since their load times are reasonable and using Chorus' filters to select which `Identifier` to view was efficient.

It may also be desirable to use Chorus to view the metadata for all times of impact. To efficiently facilitate this, the `OutputNum` variable can be noted in addition to the impact time. With this variable, the analyst can manually edit one of the `User` variables at the line number specified by `OutputNum` within the `workdir*/hscth.individual.csv` files. Loading in *hscth.combined1.csv*, determining the impact times for `Identifier` `1` through `10`, noting their `OutputNum` line numbers, and then manually editing the 10 `workdir.*/hscth.individual.csv` files required almost 16 minutes. Performing this 10 times requires about 2.7 hours to manually determine all times of impact and to edit the CSV files to note this.


Tecplot Chorus DE 2013 Beta - [Welcome]
File Edit View Project Help
Filters
Properties
Open Session... Open Project... Create Project...
Ready

Recent Sessions
workdir2.chses (combined2.chprj)
workdir1.chses (workdir1.chprj)

Recent Projects
filtered1.chprj
piece1.chprj
combined2.chprj
combined2.chprj
workdir1.chprj
workdir1.chprj
loadinexample.chprj
example1.chprj

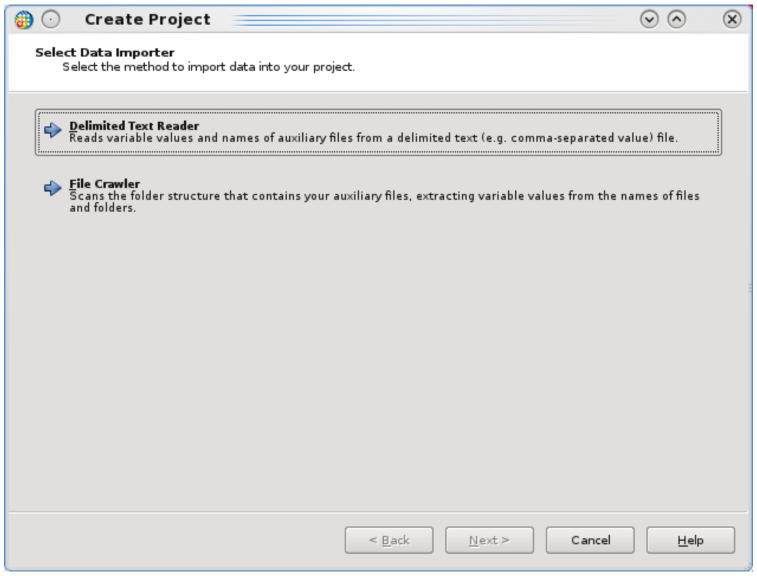
Documentation
Getting Started
User's Manual
Release Notes

Online Resources
Sorry, the online resources are not currently available. Is your Internet connection working?

Tecplot Chorus DE 2013 Beta
Version 3.1.4 (build 53675)
Linux (64-bit)

support@tecplot.com
+1 425 653 9393

Figure 3.2: Initial Tecplot Chorus window; select `[Create Project...]` if this is the 1st time data is being imported

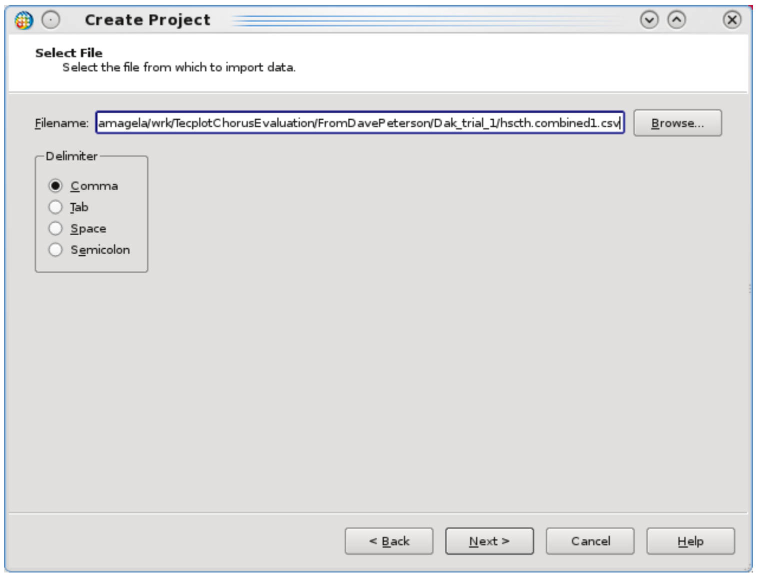


Select Data Importer
Select the method to import data into your project.

Delimited Text Reader
Reads variable values and names of auxiliary files from a delimited text (e.g. comma-separated value) file.

File Crawler
Scans the folder structure that contains your auxiliary files, extracting variable values from the names of files and folders.

Figure 3.3: Tecplot Chorus wizard window for importing data; most of the time, the `[Delimited Text Reader]` button should be pressed

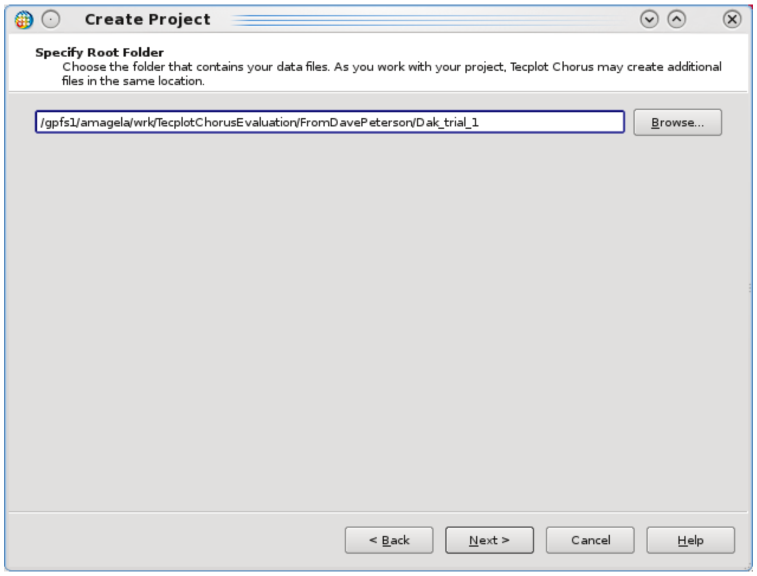


Select File
Select the file from which to import data.

Filename: amagela/wrk/TecplotChorusEvaluation/FromDavePeterson/Dak_trial_1/hscth.combined1.csv

Delimiter
(●) Comma
( ) Tab
( ) Space
( ) Semicolon

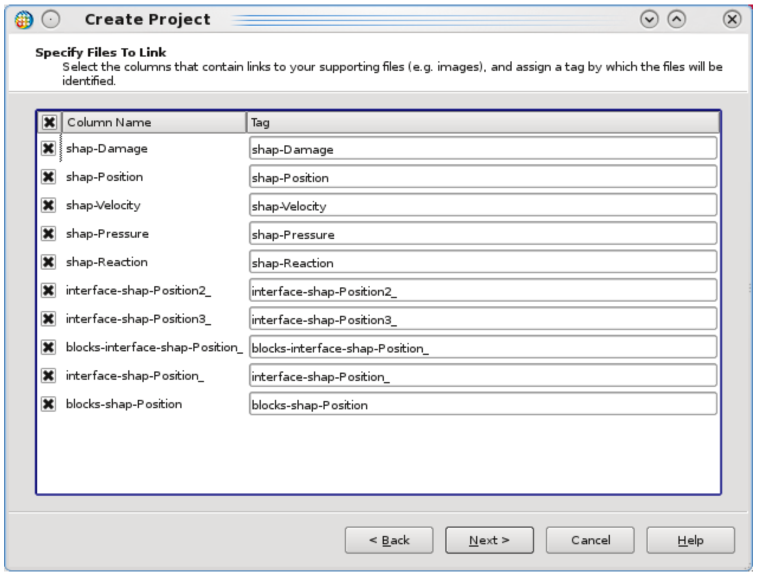
Figure 3.4: Tecplot Chorus 2nd wizard window for importing data; click on [Browse] and navigate to the CSV file to import and select the appropriate delimiter, e.g., [Comma]



Specify Root Folder
Choose the folder that contains your data files. As you work with your project, Tecplot Chorus may create additional files in the same location.

/gpfs1/amagela/wrk/TecplotChorusEvaluation/FromDavePeterson/Dak_trial_1

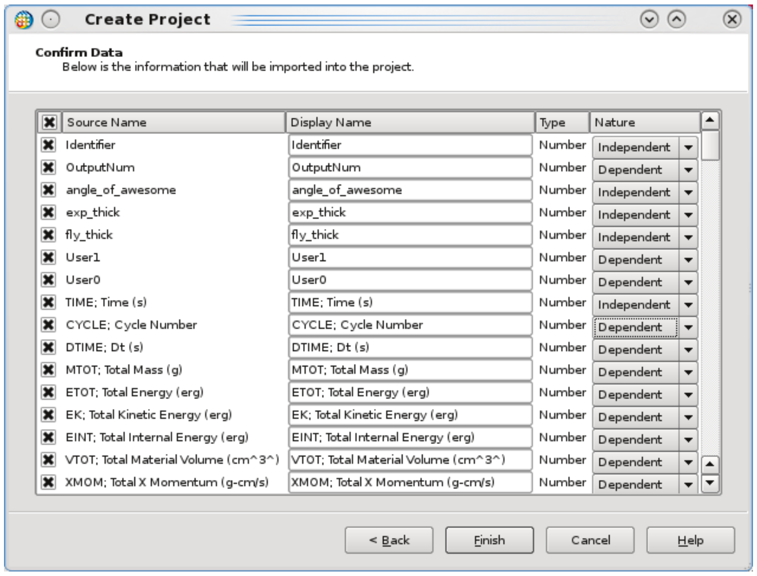
Figure 3.5: Tecplot Chorus wizard window for setting the root folder; click [Browse] and navigate to the folder where `csvcreate.py` was invoked since the image file paths are relative to this location



Specify Files To Link
Select the columns that contain links to your supporting files (e.g. images), and assign a tag by which the files will be identified.

☒	Column Name	Tag
☒	shap-Damage	shap-Damage
☒	shap-Position	shap-Position
☒	shap-Velocity	shap-Velocity
☒	shap-Pressure	shap-Pressure
☒	shap-Reaction	shap-Reaction
☒	interface-shap-Position2_	interface-shap-Position2_
☒	interface-shap-Position3_	interface-shap-Position3_
☒	blocks-interface-shap-Position_	blocks-interface-shap-Position_
☒	interface-shap-Position_	interface-shap-Position_
☒	blocks-shap-Position	blocks-shap-Position

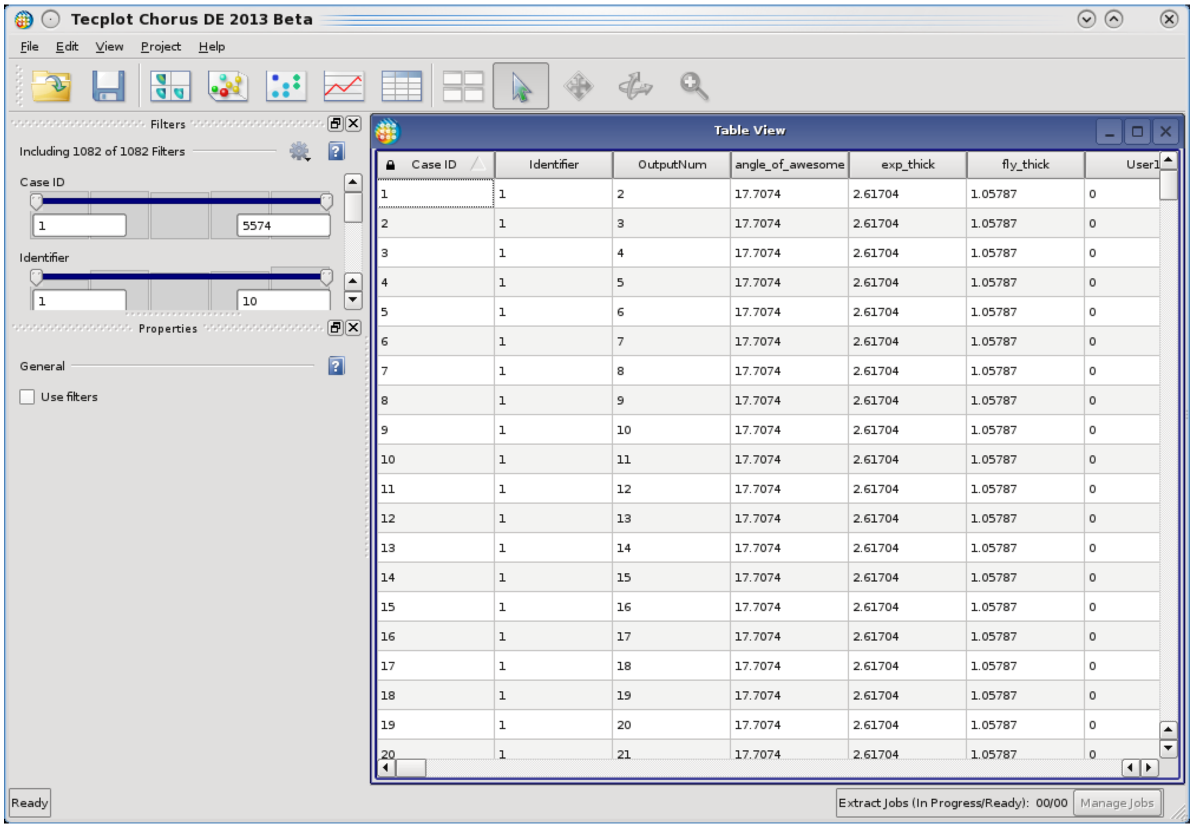
Figure 3.6: Tecplot Chorus wizard window for specifying columns containing image files; check the column names that contain image file locations and provide a ``Tag'' for their reference



Confirm Data
Below is the information that will be imported into the project.

☒	Source Name	Display Name	Type	Nature
☒	Identifier	Identifier	Number	Independent
☒	OutputNum	OutputNum	Number	Dependent
☒	angle_of_awesome	angle_of_awesome	Number	Independent
☒	exp_thick	exp_thick	Number	Independent
☒	fly_thick	fly_thick	Number	Independent
☒	User1	User1	Number	Dependent
☒	User0	User0	Number	Dependent
☒	TIME; Time (s)	TIME; Time (s)	Number	Independent
☒	CYCLE; Cycle Number	CYCLE; Cycle Number	Number	Dependent
☒	DTIME; Dt (s)	DTIME; Dt (s)	Number	Dependent
☒	MTOT; Total Mass (g)	MTOT; Total Mass (g)	Number	Dependent
☒	ETOT; Total Energy (erg)	ETOT; Total Energy (erg)	Number	Dependent
☒	EK; Total Kinetic Energy (erg)	EK; Total Kinetic Energy (erg)	Number	Dependent
☒	EINT; Total Internal Energy (erg)	EINT; Total Internal Energy (erg)	Number	Dependent
☒	VTOT; Total Material Volume (cm^3^)	VTOT; Total Material Volume (cm^3^)	Number	Dependent
☒	XMOM; Total X Momentum (g-cm/s)	XMOM; Total X Momentum (g-cm/s)	Number	Dependent

Figure 3.7: Tecplot Chorus wizard window for setting variable nature, e.g., independent, dependent, or other; for *CTH Impact Example*, ``angle_of_awesome'', ``exp_thick'', ``fly_thick'', ``Identifier'', ``TIME'', and ``CYCLE'' are the only independent variables



Including 1082 of 1082 Filters

Case ID: 1 – 5574

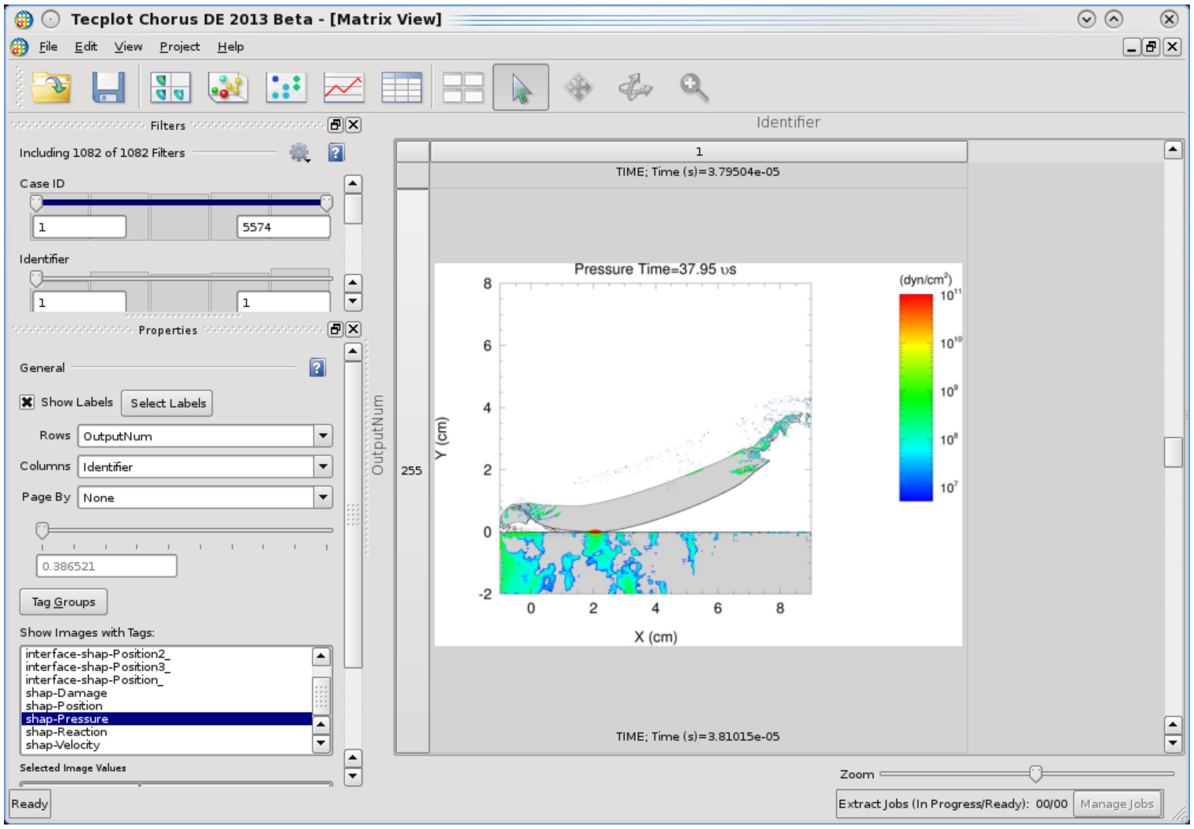
Identifier: 1 – 10

Use filters

Case ID	Identifier	OutputNum	angle_of_awesome	exp_thick	fly_thick	User1
1	1	2	17.7074	2.61704	1.05787	0
2	1	3	17.7074	2.61704	1.05787	0
3	1	4	17.7074	2.61704	1.05787	0
4	1	5	17.7074	2.61704	1.05787	0
5	1	6	17.7074	2.61704	1.05787	0
6	1	7	17.7074	2.61704	1.05787	0
7	1	8	17.7074	2.61704	1.05787	0
8	1	9	17.7074	2.61704	1.05787	0
9	1	10	17.7074	2.61704	1.05787	0
10	1	11	17.7074	2.61704	1.05787	0
11	1	12	17.7074	2.61704	1.05787	0
12	1	13	17.7074	2.61704	1.05787	0
13	1	14	17.7074	2.61704	1.05787	0
14	1	15	17.7074	2.61704	1.05787	0
15	1	16	17.7074	2.61704	1.05787	0
16	1	17	17.7074	2.61704	1.05787	0
17	1	18	17.7074	2.61704	1.05787	0
18	1	19	17.7074	2.61704	1.05787	0
19	1	20	17.7074	2.61704	1.05787	0
20	1	21	17.7074	2.61704	1.05787	0

Extract Jobs (In Progress/Ready): 00/00

Figure 3.8: Tecplot Chorus window providing Table View



Including 1082 of 1082 Filters

Case ID: 1 – 5574

Identifier: 1 – 1

General

[✓ Show Labels] Select Labels

Rows: OutputNum

Columns: Identifier

Page By: None

0.386521

Tag Groups

Show Images with Tags:

- interface-shap-Position2_
- interface-shap-Position3_
- interface-shap-Position_
- shap-Damage
- shap-Position
- [✓ shap-Pressure]
- shap-Reaction
- shap-Velocity

Selected Image Values

Identifier 1

TIME; Time (s)=3.79504e-05

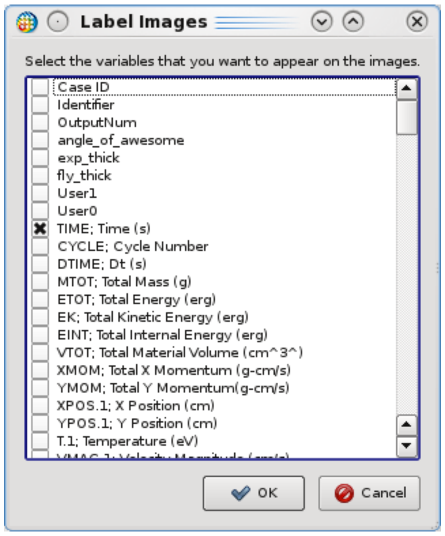
255

TIME; Time (s)=3.81015e-05

Ready

Extract Jobs (In Progress/Ready): 00/00

Figure 3.9: Tecplot Chorus window providing image Matrix View; ‘‘OutputNum’’ 255 (line number 255 of CSV file ignoring its header) at time 3.81015e-05 is the first image to exhibit impact



Select the variables that you want to appear on the images.

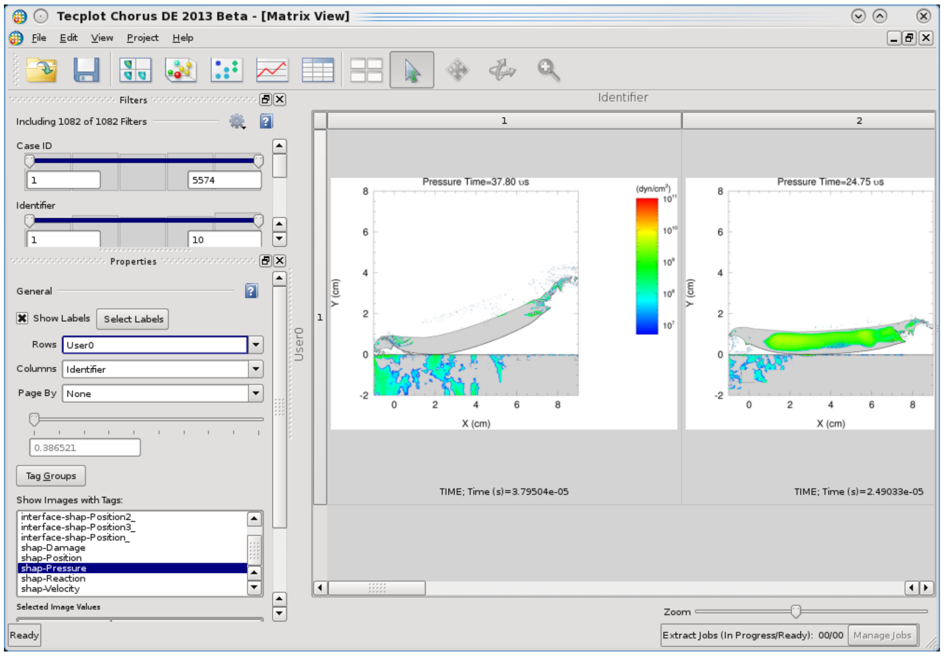
- [ ] Case ID
- [ ] Identifier
- [ ] OutputNum
- [ ] angle_of_awesome
- [ ] exp_thick
- [ ] fly_thick
- [ ] User1
- [ ] User0
- [✓ TIME; Time (s)]
- [ ] CYCLE; Cycle Number
- [ ] DTIME; Dt (s)
- [ ] MTOT; Total Mass (g)
- [ ] ETOT; Total Energy (erg)
- [ ] EK; Total Kinetic Energy (erg)
- [ ] EINT; Total Internal Energy (erg)
- [ ] VTOT; Total Material Volume (cm^3^)
- [ ] XMOM; Total X Momentum (g-cm/s)
- [ ] YMOM; Total Y Momentum(g-cm/s)
- [ ] XPOS.1; X Position (cm)
- [ ] YPOS.1; Y Position (cm)
- [ ] T.1; Temperature (eV)

Figure 3.10: Tecplot Chorus Label Images window set to label images with TIME

3.6 Workflow Stage 4: Create Filtered CSV File

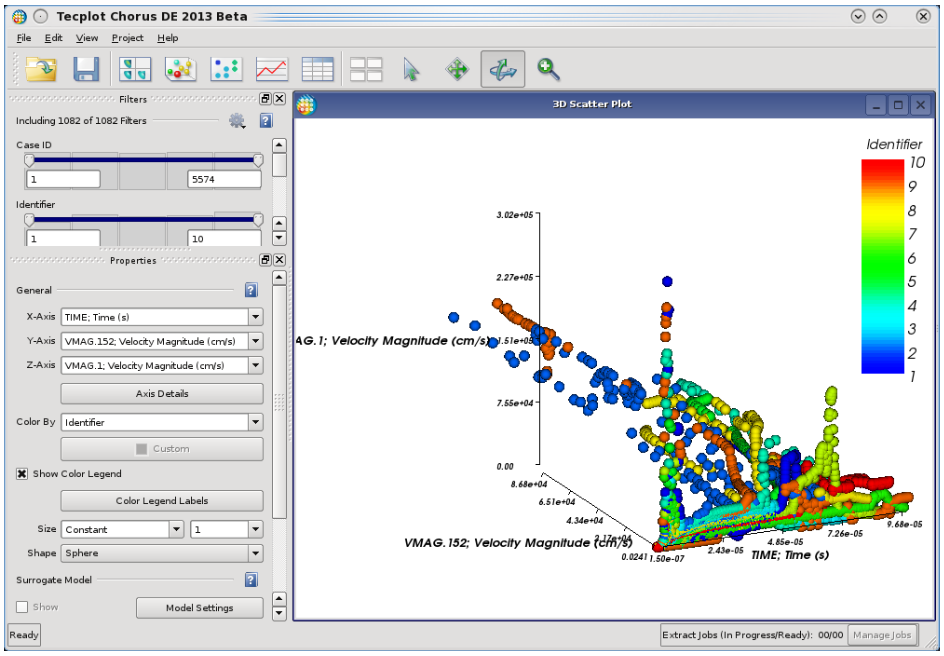
After Section 3.5, the impact times (goal 2) have been determined and noted within the `workdir*/hscth.individual.csv` files. To create an updated, combined set of CSV files, rerun `` `csvcreate.py -2` ``; now these updated `User` variables are within `hscth.combined.csv` and `hscth.combined*.csv`. To extract only those points out, execute `` `csvcreate.py -3` ``, which will quickly create a `hscth.userfiltered.csv` file. This file can be loaded into Tecplot Chorus for further investigations. Some figures that showcase more Tecplot Chorus useful views are given below in Figures 3.11-3.15.

The reduced, likely candidate solutions that have now been distilled can be further analyzed analytically or qualitatively to assess conditions such as angle of impact, final impact closure time, spallation, or any other features that are desired. This approach provides an efficient method to distill a medium to large dataset into a much more manageable grouping for study. Often, this level of effort is what is required to be capable of making a reasonable recommendation for design of testing that will follow. It provides a route that requires relatively little future user scripting and provides a tool that an analyst can quickly utilize for smaller projects where a fully automated optimization algorithm might not be required. It meets the point where time of post-processing is adequately diminished to the degree that further specialization of selection tools is unwarranted and a decision can be reached. Furthermore, if a larger field of study is then determined to be required, a more thorough parameter study can be run from the output from the initial scripts used to assemble the `hscth.combined*.csv` or `hscth.userfiltered.csv` files.



Including 1082 of 1082 Filters
Case ID: 1 – 5574
Identifier: 1 – 10
Show Labels [✓]
Rows: User0
Columns: Identifier
Page By: None
0.386521
Show Images with Tags:
interface-shap-Position2_
interface-shap-Position3_
interface-shap-Position_
shap-Damage
shap-Position
[✓ shap-Pressure]
shap-Reaction
shap-Velocity
Selected Image Values
TIME; Time (s)=3.79504e-05
TIME; Time (s)=2.49033e-05
Extract Jobs (In Progress/Ready): 00/00

Figure 3.11: Tecplot Chorus Matrix view showing side-by-side pressure plots at impact



Filters

Including 1082 of 1082 Filters

Case ID: 1 – 5574

Identifier: 1 – 10

Properties

General

X-Axis: TIME; Time (s)

Y-Axis: VMAG.152; Velocity Magnitude (cm/s)

Z-Axis: VMAG.1; Velocity Magnitude (cm/s)

Axis Details

Color By: Identifier

Custom

[✓] Show Color Legend

Color Legend Labels

Size: Constant, 1

Shape: Sphere

Surrogate Model

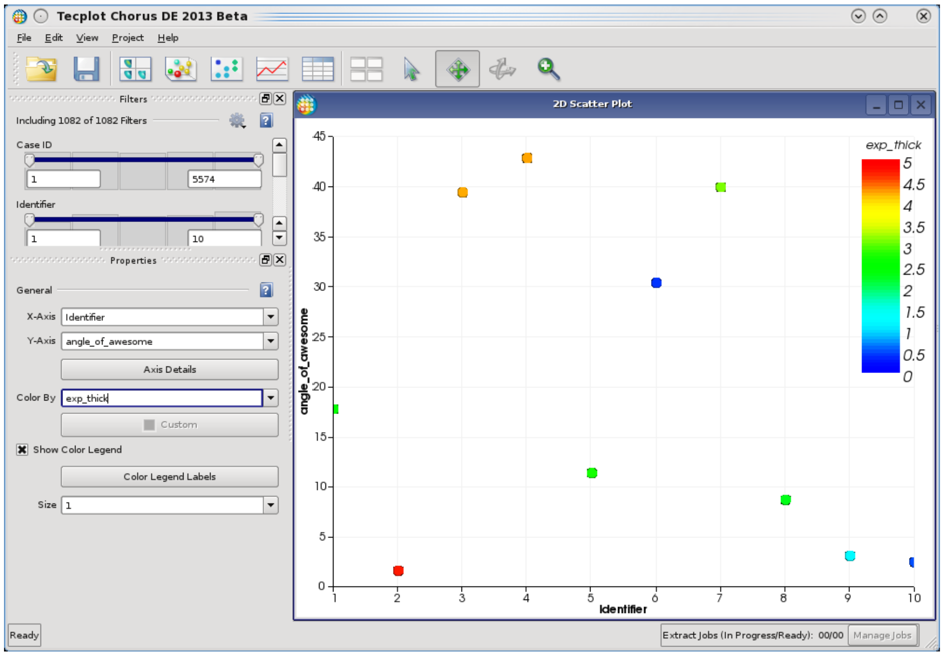
[ ] Show

Model Settings

Ready

Extract Jobs (In Progress/Ready): 00/00 Manage Jobs

Figure 3.12: Tecplot Chorus 3D Scatter Plot view showing temporal trend of velocity magnitudes at different locations colored by Dakota seed number



Filters

Including 1082 of 1082 Filters

Case ID: 1 – 5574

Identifier: 1 – 10

Properties

General

X-Axis: Identifier

Y-Axis: angle_of_awesome

Axis Details

Color By: exp_thick

Custom

[✓] Show Color Legend

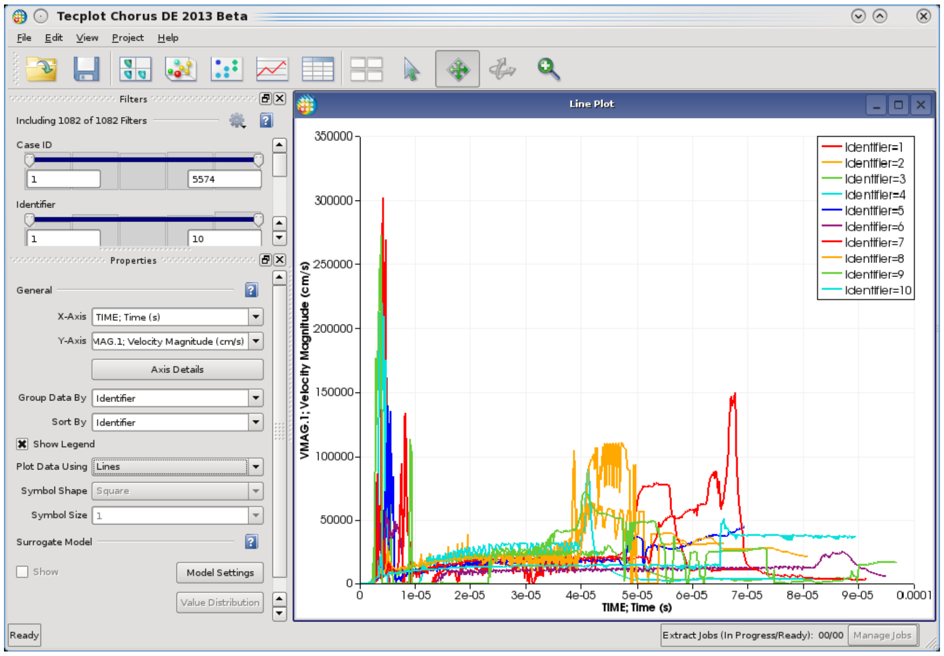
Color Legend Labels

Size: 1

Ready

Extract Jobs (In Progress/Ready): 00/00 Manage Jobs

Figure 3.13: Tecplot Chorus 2D Scatter Plot view showing the reference angle as a function of Dakota seed number and colored by explosive thickness



Including 1082 of 1082 Filters

Case ID: 1 – 5574

Identifier: 1 – 10

General

X-Axis: TIME; Time (s)

Y-Axis: MAG.1; Velocity Magnitude (cm/s)

Group Data By: Identifier

Sort By: Identifier

☒ Show Legend

Plot Data Using: Lines

Symbol Shape: Square

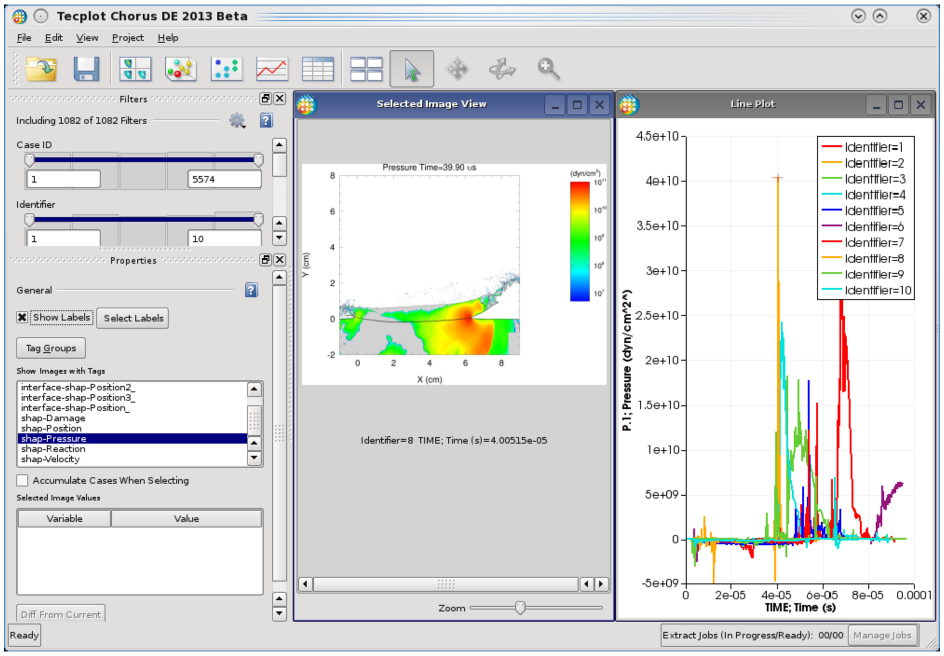
Symbol Size: 1

Surrogate Model

☐ Show

Extract Jobs (In Progress/Ready): 00/00

Figure 3.14: Tecplot Chorus Line Plot view showing the velocity magnitude of tracer 1 as a function of time grouped by Dakota seed number



Including 1082 of 1082 Filters

Case ID: 1 – 5574

Identifier: 1 – 10

General

☒ Show Labels

Show Images with Tags

interface-shap-Position2_
interface-shap-Position3_
interface-shap-Position_
shap-Damage
shap-Position
shap-Pressure
shap-Reaction
shap-Velocity

☐ Accumulate Cases When Selecting

Selected Image Values

Variable	Value

Identifier=8 TIME; Time (s)=4.00515e-05

Extract Jobs (In Progress/Ready): 00/00

Figure 3.15: Tecplot Chorus Selected Image view; a point within the Line Plot window is selected and the figure is displayed within the Selected Image window

Chapter 4

Conclusions and Future Work

Analyzing ensemble data has many challenges, some of which are listed below.

Portability Ensemble data is expected to have a large file system footprint with many files. Moving or copying this data may not be practical.

Data Locality Ensemble data will likely be generated on local and remote HPC platforms. How and where the data are processed will be a function of tool and security requirements along with ensemble portability.

Processing Scalability As the number of simulations within an ensemble increase, so does the time to process them individually and as a whole.

Tools need to be used and developed to assist with these growing challenges. One of these tools is Tecplot Chorus, a tool for visualizing ensemble metadata and images, deriving quantities from the metadata, and interfacing with other tools, e.g., Tecplot 360 to create additional images to view. Chorus enabled an analysis of *CTH Impact Example* to visually determine points of impact in an efficient manner. However, this does come with caveats, some of which are listed below.

1. Chorus can only see data on the system it is running on.
2. To use Chorus on a local computer, it would have to be running on the local computer with the ensemble data present or it would have to run on the platform with the data and be displayed on the local computer by X11 forwarding or screen capture technology, e.g., VNC, NX, RGS.
3. Chorus will load all of the ensemble metadata provided to it into memory and will only load images upon request. Chorus' RAM requirements will grow as the metadata sizes grow. The *CTH Impact Example* monolithic CSV file is approximately 753 MB. When Chorus loads this into memory, it is using upwards of 26 GB of RAM, which is more than what is available on most laptop and desktop systems. Moreover Chorus is unable to properly save its own Project and Session files on a dataset this large since it wants to save out a file that is larger than 2 GB and it was unable to; 32-bit issues within the code are the likely culprit.

4. Aside from filtering, Chorus does not have many analysis capabilities to help extract quantities of interest.

Despite the caveats above, Chorus does have some compelling strengths, some of which are listed below.

1. Chorus provides many views to visualize metadata; these views are Table View, Matrix View (looking at a matrix of images), 3D Scatter Plot, 2D Scatter Plot, Line Plot, and the ability to efficiently tile many open views in an organized fashion.
2. Chorus has easy-to-understand filtering with the sliders on the right side of their window that allow the analyst to select which ranges remain in all of the Views.
3. Chorus can create surrogate models of the data, which can be used to visually interpolate, extrapolate, or export to other tools such as MATLAB for further processing.
4. Chorus can create new variables on the fly based on simple arithmetic operations based on existing variables.
5. If the simulations write out their data in file formats compatible with Tecplot 360, Chorus can leverage 360 to create new views that are defined from a single example and it will create those views for all other members of the ensemble in a batch sense.

If the analysis of interest has needs that align with Chorus' strengths, then it will provide a lot of value, assuming its shortcomings can be overcome with tools such as `csvcreate.py` that can provide multiple scales of data to be loaded into Chorus. The timing information from the evaluation performed in Chapter 3 and the Appendices is below within Table 4.1.

Table 4.1: Computational and user time spent performing evaluation of *CTH Impact Example* with Tecplot Chorus

Task	**Computer Time (min.)**	**Individual Time (min.)**	**Elapsed Time (min.)**
Workflow 1	228.7	-	228.7
Workflow 2	17.1	-	245.8
Workflow 3	1.3	-	247.1
Visually Determine Impact	-	158.4	405.5
Workflow 4	0.2	-	405.7
Totals	*247.3*	*158.4*	*405.7*

References

[1] sandialabs/slycat: Web-based science analysis and visualization platform. https://github.com/sandialabs/slycat, September 2014.

[2] Tecplot Mission and Values. http://www.tecplot.com/the-company, September 2014.

[3] Tecplot Chorus - CFD Post processor, data manager, analytics tool. http://www.tecplot.com/products/tecplot-chorus, September 2014.

[4] Tecplot 360 CFD post processing software to Analyze complex data. http://www.tecplot.com/products/tecplot-360, September 2014.

[5] Tecplot Chorus DE - Simulation Analytics for Making Better Decisions Faster. http://download.tecplot.com/docs/Tecplot_Chorus_Datasheet.pdf, September 2014.

[6] B. M. Adams, M. S. Ebeida, M. S. Eldred, J. D. Jakeman, L. P. Swiler, J. A. Stephens, D. M. Vigil, T. M. Wildey, W. J. Bohnhoff, K. R. Dalbey, J. P. Eddy, K. T. Hu, L. E. Bauman, and P. D. Hough. DAKOTA, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.0 User's Manual. Technical report SAND2014-4633, Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550, July 2014.

[7] Sandia National Laboratories: CTH Shock Physics: Home. http://www.sandia.gov/CTH/index.html, September 2014.

[8] ImageMagick: Convert, Edit, Or Compose Bitmap Images. http://www.imagemagick.org, September 2014.

[9] Python Distribution and Integrated Analysis Environment | Enthought Canopy. https://www.enthought.com/products/canopy/, December 2014.

Appendix A

Ensemble Workflow Stage 1 with `csvcreate.py`

This appendix discusses methods and motivations for batch processing image files. Analyzing a plethora of images can be significantly slowed if the images themselves are not adequately compressed. The *CTH Impact Example* contains a total of 534,240 Spymaster JPEG files with a resolution of `3840x2160` pixels. These files use 237.1 GB of storage space. This section provides an investigation of scalable methods for reducing the image storage requirements, which will increase their portability and responsiveness with viewing.

The intended methodology for viewing the images is with Tecplot Chorus. A screenshot of an image viewed within Tecplot Chorus from the *CTH Impact Example* is in Figure A.1. The following enumerated list contains possible image operations that, after examining Figure A.1, may reduce the image sizes.

1. The images from Spymaster are saved within a JPEG format. Other formats, e.g., PNG, may require less storage space for the same image quality.
2. The image displayed within Tecplot Chorus in Figure A.1 contains a lot of unused white space. Cropping the image and removing these unused pixels may help reduce the image size. The Chorus window in Figure A.1 is maximized to use the full monitor's screen real estate and the image within is at the maximum zoom level that Chorus will allow.
3. The image's standard resolution is `3840x2160`. The resolution of the entire Tecplot Chorus window, which also encapsulates the image, is `1920x1142`. This implies the image as displayed on the screen is approximately `800x450` or 360,000 pixels, which is significantly smaller than its standard resolution. Scaling the image to the maximum size that Tecplot Chorus can accommodate may help reduce the image file size.

The image operations enumerated above are straightforward but would require a significant amount of time to manually perform, therefore tools that can perform these operations en masse are required. ImageMagick[8] is a suite of tools that meets this requirement and

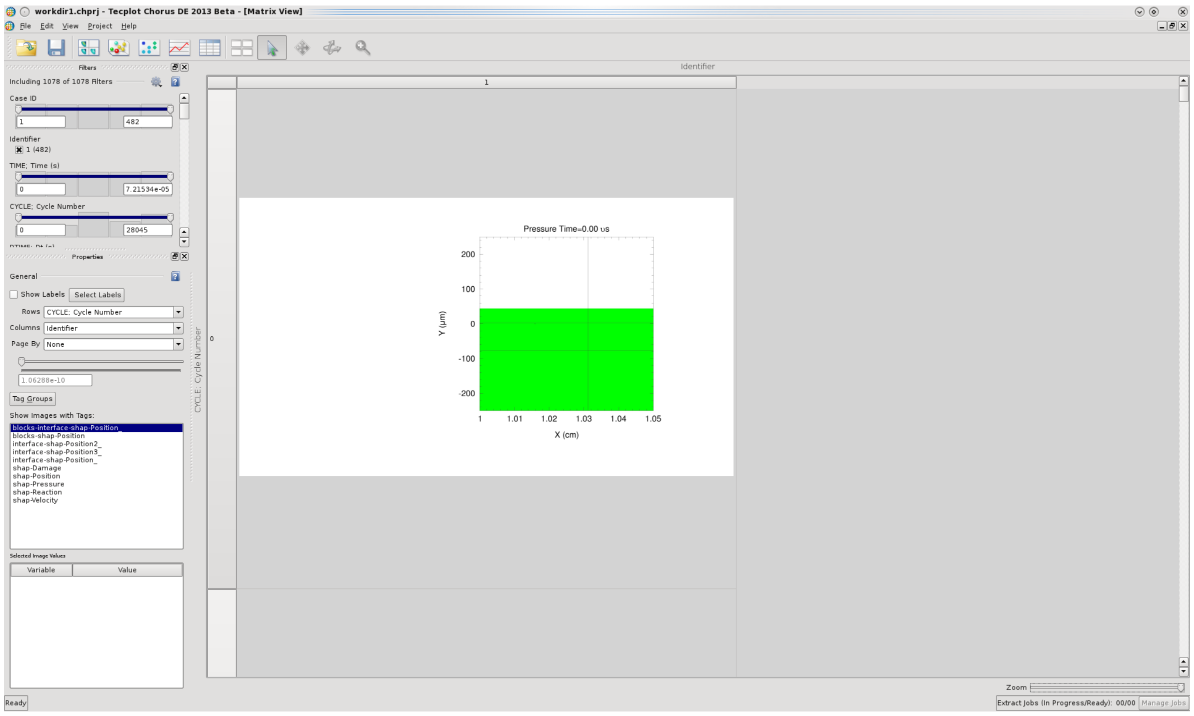


Figure A.1: Tecplot Chorus maximized and displaying an image with maximum zoom

is already installed on most HPC platforms, so it was used to assess the impact of the aforementioned image operations.

The image operations are scripted within `csvcreate.py` and can be invoked by executing `` `csvcreate.py -O` ``. When that command is invoked, `csvcreate.py` will fork a sub-process that executes a GNU BASH script that actually performs the operations. To make changes to the image operations, first execute `` `csvcreate.py -O -w` `` which will create a script in the current directory named `fixpics.sh`. Then, make the necessary changes to `fixpics.sh` and then execute `` `csvcreate.py -O` ``, which will pick up the `fixpics.sh` script within the directory and use that.

Table A.1 shows the impact each operation from executing `` `csvcreate.py -O` `` has on the `workdir.1/*000000.jpg` files from the *CTH Impact Example*.

Table A.1: File sizes of `workdir.1/*000000.jpg` as image operations are performed

Operation	**Size**	**Individual Reduction**	**Total Reduction**
Default	3,888 KB		0.0%
Convert from JPEG to PNG	1,200 KB	69.1%	69.1%
Crop and remove white space	1,008 KB	16.0%	74.1%
Scale to Chorus-relevant resolution	560 KB	44.4%	85.6%

Each of the image operations was shown to reduce file sizes. Cumulatively, the operations reduced the file sizes of `workdir.1/*000000.jpg` by 85.6%. The output from executing `` `csvcreate.py -O` `` on `cee-build005` for all 534,240 JPEG files is shown below. Overall, `csvcreate.py` performed those operations on all files within 3.8 hours. During that time frame, approximately 40 images were processed per second. The 534,240 JPEG files consume 237.1 GB of space; the resultant PNG files created from the script consume 31.3 GB, which is only 13.2% of the size of the initial JPEG files.

This portion of `csvcreate.py` is threaded. The ImageMagick program being called is named `convert`, which itself is threaded; the number of threads it spawns is limited within `csvcreate.py` by the variable `MAGICK_THREAD_LIMIT`. In addition to that, `csvcreate.py` will create its own threads based on the JPEG files it finds; the maximum number of these threads is limited within `csvcreate.py` by the variable `NumThreads`. The default values within `csvcreate.py` are sensible, but may need to be adjusted depending on the system and its load.

The Chorus-relevant resolution that `csvcreate.py` targets is the image resolution that maintains the source image's aspect ratio while targeting 360,000 total pixels. This cumulative value of pixels was used based on the observation above that only `800x450` of an image was being displayed.

```
$ ./csvcreate.py -O
csvcreate.py INFO: Did not find image formatting script; will use built-in script.
INFO: Will create 64 threads at a time
No. of files processed (displayed every 64 files): 534208
INFO: Finished with the following timing information:

real    228m42.588s
user    22453m51.787s
sys     677m15.662s
csvcreate.py INFO: These operations required 228.710247064 minutes to complete.
```

Output from executing `` `csvcreate.py -O` ``

```
#This script will crop and resize Spymaster output from
#CTH example with ImageMagick.

export MAGICK_THREAD_LIMIT=8

#GOOD PERFORMANCE BY SETTING NUMBER OF THREADS EQUAL TO 2X AVAILABLE
if test -f /proc/cpuinfo ; then
    NumThreads=$( grep processor /proc/cpuinfo | wc -l )
    NumThreads=${NumThreads:-8}
    NumThreads=$((NumThreads * 2))
else
    NumThreads=8
fi
if test $NumThreads -gt 64 ; then
```

```
    NumThreads=64
fi
echo "INFO: Will create $NumThreads threads at a time"

convertme ()
{
    if test ! -f "${1%.jpg}".png ; then
        convert "${1}" -trim +repage -resize 360000@ -gravity center "${1%.jpg}".png
    fi
}
export -f convertme

main ()
{
    declare -i i
    declare -i j
    i=0
    j=0
    find . -type f -name "*.jpg" |
    while read file ; do
        if test $i -ge $NumThreads ; then
            echo -ne "No. of files processed (displayed every $NumThreads files): ${j}"
            wait
            i=0
        fi
        convertme "${file}" &
        ((i++))
        ((j++))
        #echo "Thread number: ${i}"
        #echo -ne "Number of files processed: ${j}"
    done
    wait
    echo
    echo "INFO: Finished with the following timing information:"
}
export -f main

time main

exit 0
```

Threaded script to crop, scale, and convert Spymaster output to PNG with ImageMagick; script is generated from the command `csvcreate.py -O -w`

Appendix B

Ensemble Workflow Stage 2 with `csvcreate.py`

The 2nd stage of the *Ensemble Workflow* is to create a database for each seed within the ensemble. This stage allows for seed-specific processing and for easy seed-based combining into larger databases. Most tools are able to export and import CSV files, which make it a convenient interface for ensemble data. The *CTH Impact Example* contains CTH `hscth` files, which are CSV files, for each directory. Additionally, each directory contains a Dakota `params.in` file, which lists all of the parameters Dakota is changing. Finally, each directory also contains Spymaster JPEG output that occurs at a smaller frequency than the `hscth` output. The following list encapsulates the goals for the CSV files to be created for each directory. These goals are also accomplished with `csvcreate.py`; the sub-bullets below list relevant `csvcreate.py` notes.

- It is desired to be able to view CTH `hscth` files for each directory.
  - ◇ The standard `hscth` files contain a 3-line header whereas most CSV readers prefer a single-line header, so `csvcreate.py` ignores the 1st line, which just lists some version and date information, and combines the 2nd and 3rd lines separated by a semicolon.
- It is desired to be able to view CTH `hscth` files that are correlated with available Spymaster image output.
  - ◇ Currently, Spymaster image file names end with a zero-padded, six-digit cycle number and a `jpg` extension, so `csvcreate.py` finds directories that contain `hscth` files and, within those directories, looks for Spymaster image file names that have the aforementioned characteristics
  - ◇ The `-f` command line flag to `csvcreate.py` allows selecting either JPEG or PNG (refer to Appendix A for how PNG files were created) to search for.
  - ◇ Once `csvcreate.py` has found all `hscth` and Spymaster image files, it will read the cycle numbers from `hscth` (column number 2) and from the Spymaster image file names and only output the row of data that corresponds to a cycle number present with both `hscth` and the image file.

        * This methodology was used to create reduced-size CSV files with no blank entries. Alternatives include listing all of the contents of the `hscth` file and leaving the Spymaster image entries blank where applicable or, instead of blank placeholders, using the images from the nearest available cycle number. Either of these alternatives would be trivial to add to `csvcreate.py`.
    - ◇ The image file column names correspond to their names that precede the cycle number.
    - ◇ The image file names are relative paths from where `csvcreate.py` was invoked.

- It is desired to add available Dakota `params.in` metadata to the resultant CSV files.
    - ◇ Each time `csvcreate.py` finds a `hscth` file within a directory, it will then see if that same directory contains a Dakota `params.in` file and, if it does, the Dakota inputs listed within this file are automatically added to that directory's resultant CSV file.

- It is desired to add additional user variables (for manual editing) to the resultant CSV files.
    - ◇ By default, `csvcreate.py` will add 1 user variable (named `User1`) to the resultant CSV files.
    - ◇ The command line flag `-n` allows the user to change this number; currently, integer values between and including 0 and 8 are permitted.

The output from executing `csvcreate.py` to create 2 user variable and use PNG output instead of JPEG is given below.

```
$ ./csvcreate.py -1 -f PNG -n 2
csvcreate.py INFO: I will recursively walk this directory (/gpfs1/amagela/wrk/
    TecplotChorusEvaluation/FromDavePeterson/Dak_trial_1) and find CTH hscth, CTH
    Spymaster PNG images, and Dakota params.in files and correlate them.
csvcreate.py INFO: I found 301 directories.
csvcreate.py INFO: I found 1095003 files.
csvcreate.py INFO: I found hscth within /gpfs1/amagela/wrk/TecplotChorusEvaluation/
    FromDavePeterson/Dak_trial_1/workdir.1.
csvcreate.py INFO: I found 4820 PNG files.
csvcreate.py INFO: Reading /gpfs1/amagela/wrk/TecplotChorusEvaluation/FromDavePeterson/
    Dak_trial_1/workdir.1/params.in
csvcreate.py INFO: Reading /gpfs1/amagela/wrk/TecplotChorusEvaluation/FromDavePeterson/
    Dak_trial_1/workdir.1/hscth
csvcreate.py INFO: Writing /gpfs1/amagela/wrk/TecplotChorusEvaluation/FromDavePeterson/
    Dak_trial_1/workdir.1/hscth.individual.csv
csvcreate.py INFO: There will be 1091 columns in resultant CSV file
#####<REPEAT LAST 5 LINES (WITH CHANGING NO. OF PNG FILES) FOR 98 OTHER WORKDIR
    DIRECTORIES>#####
csvcreate.py INFO: These operations required 17.1167577346 minutes to complete.
```

Output from executing `csvcreate.py -1 -f PNG -n 2`

Appendix C

Ensemble Workflow Stage 3 with csvcreate.py

The 3rd stage of the *Ensemble Workflow* is to create a database for the entire ensemble. This stage creates a monolithic ensemble database that can either be processed with other tools or used as a single point to further filter. The creation of this monolithic database is handled within `csvcreate.py` with the `-2` command line argument. This argument will also create "piecemeal" files that are the combination of a specified number of individual `hscth.individual.csv` files. These "piecemeal" files, named `hscth.combined#.csv` where `#` is an integer listing the piece number, allow loading in smaller pieces for more efficient processing, if needed.

When `` `csvcreate.py -2` `` is invoked, it forks a sub-process that runs a GNU BASH script to perform the concatenation. This script can be exported to the current directory with the command `` `csvcreate.py -2 -w` `` and it will be named `combinecsv.sh`; this script is replicated at the end of this Appendix for reference. This script can be edited and, when `` `csvcreate.py -2` `` is executed again, it will be run. The variable to control how many `hscth.individual.csv` files are put into each "piecemeal" one is named `NUMCSV`.

Sample output from executing these scripts is given below.

```
$ ./csvcreate.py -2
csvcreate.py INFO: Did not find CSV concatenation script (combinecsv.sh); will use built-
    in script.
INFO: Creating single, combined CSV file hscth.combined.csv
INFO: Creating piecemeal CSV file (hscth.combined1.csv)
INFO: Creating piecemeal CSV file (hscth.combined2.csv)
INFO: Creating piecemeal CSV file (hscth.combined3.csv)
INFO: Creating piecemeal CSV file (hscth.combined4.csv)
INFO: Creating piecemeal CSV file (hscth.combined5.csv)
INFO: Creating piecemeal CSV file (hscth.combined6.csv)
INFO: Creating piecemeal CSV file (hscth.combined7.csv)
INFO: Creating piecemeal CSV file (hscth.combined8.csv)
INFO: Creating piecemeal CSV file (hscth.combined9.csv)
INFO: Creating piecemeal CSV file (hscth.combined10.csv)
INFO: Finished after 78 seconds
csvcreate.py INFO: These operations required 1.29485498269 minutes to complete.
```

Output from executing `csvcreate.py -2`

```bash
#This defines the name of the combined CSV file
NEWFILE=hscth.combined.csv

#This defines the name of the CSV files to search for
INFILE=hscth.individual.csv

#This defines how many CSV files to put into a piecemeal CSV file
NUMCSV=10

echo "INFO: Creating single, combined CSV file ${NEWFILE}"
HEADER="$( find . -name $INFILE -print -quit )"
awk 'NR==1' $HEADER > $NEWFILE
find . -type f -name $INFILE -print0 | xargs -0 -I file awk 'NR>1' file >>$NEWFILE

NEWFILE=${NEWFILE%.csv}
inmcsv=$NUMCSV
((inmcsv++))
jnmcsv=0
find . -type f -name $INFILE | sort -V |
while read file ; do
    ((inmcsv++))
    fileout="${NEWFILE}${jnmcsv}.csv"
    if test $inmcsv -gt $NUMCSV ; then
        inmcsv=1
        ((jnmcsv++))
        fileout="${NEWFILE}${jnmcsv}.csv"
        echo "INFO: Creating piecemeal CSV file (${fileout})"
        awk 'NR==1' $HEADER > $fileout
    fi
    awk 'NR>2' "${file}" >> $fileout
done

echo "INFO: Finished after $SECONDS seconds"
```

Script to combine CSV files generated from `csvcreate.py -2 -w` and named `combinecsv.sh`

Appendix D

Ensemble Workflow Stage 4 with `csvcreate.py`

The 4th *Ensemble Workflow* stage is to reduce the entire ensemble database using problem-specific filters. This stage will reduce the monolithic ensemble database into something that is manageable by the tools used for processing and for the analysts to comprehend. For the *CTH Impact Example*, one of the `User` variables was manually edited to reflect the impact time. A filter was created within `csvcreate.py` to only export these nonzero `User` variables. To create a filtered CSV file (named `hscth.userfiltered.csv`) from the combined CSV file (named `hscth.combined.csv` and created within Appendix C), simply execute `` `csvcombine.py -3` ``. Example output from executing this script is shown below.

```
$ ./csvcreate.py -3
csvcreate.py INFO: These operations required 0.181731800238 minutes to complete.
```

Output from executing `` `csvcreate.py -3` ``

Appendix E

Notes and Reference for `csvcreate.py`

The full source code and a list of notes for `csvcreate.py` are given below.

- Pass `-h` or `--help` to `csvcreate.py` to view its help page.
- There are many dependencies to execute `csvcreate.py`. All of them are met with Enthought's Python 2.7 distribution[9] that is installed on the CEE LAN and on most SNL HPC platforms.
- This script supports creating CSV files that are compatible with both Chorus and Slycat; please use the `-t` or `--tool` command line options and specify either `Slycat` or `Chorus` to select which tool to target.
- This tool was written solely for this evaluation and not for production use.

```
#!/usr/bin/env python
#VERSION 2.0|20141020
#On CEE, do module load apps/epd to get pre-installed Python 2.7
#On HPC, do module load canopy   to get pre-installed Python 2.7

### PREAMBLE ##############################
import argparse   #parse command line options
import sys,os     #handle "system" and "operating system" information
import random     #create nice tags if needed
import datetime   #create nice tags if needed
import subprocess #run external commands
import urllib     #download files from internet
import time       #understand how long program takes to run
import re         #regex searching
import csv        #reading and writing CSV files
from itertools import repeat #Initialize list of lists
import tempfile   #creating temporary file

start_time = time.time()
CTHCycleDigits = 6      #CTH Spymaster JPEG files are output with this fixed-
                        # width field for cycle number
DEFIMGFORMAT="JPEG"     #DEFAULT file format to search for
```

```
DEFNUMUSER=1             #DEFAULT number of user variables to add to CSV file
DEFINDCSVFILE=".individual.csv" #MANUALLY CHANGE WITHIN BASH SCRIPT BELOW
DEFTOOL="Slycat"         #DEFAULT ensemble processing tool
DEFSRV="lynx"            #DEFAULT server for accessing files
csvcombined=’hscth.combined.csv’
csvfiltered=’hscth.userfiltered.csv’

### DEFINE COMMAND LINE OPTIONS ###########
ThisDescription = "This program creates CSV files for ensembles of simulations. It will recursively walk the directory where it is invoked and export the new CSV file there."
ThisEpilog = "This program requires Python version 2.7 (‘module load apps/epd‘ on CEE LAN, ‘module load canopy‘ on SNL HPC platforms) and is only tested on UNIX-compatible systems."
parser = argparse.ArgumentParser(
    description = ThisDescription,
    epilog = ThisEpilog,
    formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument("-v", "--version", action="version", version="%(prog)s 2.0")
parser.add_argument("-0", "--fixpics",
                    help="execute local fixpics.sh script to process image files; this is useful so multiple arguments can be set for processing to occur afterwards",
                    action="store_true")
parser.add_argument("-1", "--hscth",
                    help="correlate CTH hscth with available JPEG and params.in files and create hscth"+DEFINDCSVFILE+" files",
                    action="store_true")
parser.add_argument("-2", "--combinecsv",
                    help="combine correlated CSV files (hscth"+DEFINDCSVFILE+") into single ("+csvcombined+") and piecemeal ones",
                    action="store_true")
parser.add_argument("-3", "--filtercsv",
                    help="extract rows from single CSV file ("+csvcombined+") containing nonzero user variables into a filtered CSV file ("+csvfiltered+")",
                    action="store_true")
parser.add_argument("-t", "--tool",
                    type=str,
                    default=DEFTOOL,
                    choices=[’Slycat’,’Chorus’],
                    help="Ensemble processing tool that will read in resultant CSV file(s)")
parser.add_argument("-f", "--formatImage",
                    type=str,
                    default=DEFIMGFORMAT,
                    choices=[’JPEG’,’PNG’],
                    help="Image format to search for")
parser.add_argument("-n", "--numUser",
                    type=int,
                    default=DEFNUMUSER,
                    choices=range(0,9),
                    help="Number of user variables to add to CSV file")
parser.add_argument("-s", "--server",
                    type=str,
                    default=DEFSRV,
                    help="Server hostname to be used for absolute file references")
parser.add_argument("-w", "--writeScripts",
                    help="Write the external scripts, when applicable, for the relevant
    sections instead of executing; this is useful to override their default behavior",
                    action="store_true")
args = parser.parse_args()


### DEFINE VARS AND TAGS ##################
ThisProg = os.path.basename(__file__)
#ThisDir = os.path.realpath(os.path.dirname(__file__))
ThisDir = os.getcwd()
ThisTime = datetime.datetime.now()
ThisRand = random.randint(10,99)
ThisTag = "_"+ThisTime.strftime('%Y%m%d%H%M%S')+str(ThisRand)
#print "{} INFO: The tag used for preserving files and directories is {}.".format(
    ThisProg,ThisTag)
UserHome = os.path.expanduser("~")
#print "{} INFO: The user's HOME directory is {}.".format(ThisProg,UserHome)



### CREATE DEFINITIONS ####################
# This returns a sorted list of found items from a list
def regexList(mylist,regex):
    result = []
    for l in mylist:
        match = re.search(regex,l)
        if match:
            result += [match.group(0)]
    result.sort()
    return result

# This reads in a Dakota params.in file if present
def paramsin(mydir):
    myparamsin=mydir+'/params.in'
    result = [[],[]]
    if os.path.isfile(myparamsin):
        print "{} INFO: Reading {}".format(ThisProg,myparamsin)
        with open(myparamsin,'rb') as params:
            allparams = params.readlines()
            params.close()
        numvars = allparams[0].split()
        numvars = int(numvars[3])
        for myline in range(1,1+numvars):
            varval = allparams[myline].split()
            result[0].append(varval[1])
            result[1].append(varval[3])
        #print "{}".format(result[0])
        #print "{}".format(result[1])
    return result


# This returns a tuple containing the following 2 lists:
```

```
# 1. A list of lists of the JPEG file names: [family][cycle]
# 2. A list of lists of the JPEG file names' cycle numbers: [family][cycle]
# 3. A list of the unique family names
def spyParse(mylist):
    uniq = []
    for l in mylist:
        lpre = l.rsplit('.',1)
        lpre = lpre[0]
        lpre = lpre[0:len(lpre)-CTHCycleDigits]
        uniq.append(lpre)
        uniq = list(set(uniq))
#   print "{} INFO: Image families found are: {}".format(ThisProg,uniq)
    resultjpg   = [[] for i in repeat(None,len(uniq))] #Initialize list of lists
    resultcycle = [[] for i in repeat(None,len(uniq))] #Initialize list of lists
    for l in mylist:
        lpre = l.rsplit('.',1)
        lpre = lpre[0]
        lpre = lpre[0:len(lpre)-CTHCycleDigits]
        li = uniq.index(lpre)
        resultjpg[li].append(l)
    li = 0
    for l in resultjpg:
#       print "{} INFO: Initial JPEG family entry: {}".format(ThisProg,l[0])
        for m in l:
            mi = m.rsplit('.',1)
            mi = mi[0]
            mi = mi[len(mi)-CTHCycleDigits:len(mi)]
            mi = int(mi)
            resultcycle[li].append(mi)
        li += 1
#   for l in resultcycle:
#       print "{} INFO: Second iteration entry: {}".format(ThisProg,l[1])
    return (resultjpg,resultcycle,uniq)



# This function reads in HSCTH files and correlates them with other JPEG files
# HSCTH file has following row description:
#   1: %Spymaster version information
#   2: Variable Names
#   3: Variable Descriptions
#   n: Data (second column is CYCLE)
def hscthmod(resultjpg,resultcycle,uniq,hscthin):
    onlyprintmax=1
    onlyprinti=0
    myfulldir = os.path.dirname(hscthin)
    myreldir = os.path.relpath(myfulldir,ThisDir)
    mydir = os.path.basename(os.path.dirname(hscthin))
    myparamsin = paramsin(myfulldir)
    regexdir=re.compile("^workdir\.\d+$")
    match = re.search(regexdir,mydir)
    if match:
        mydir = mydir.rsplit('.',1)
        mydir = mydir[len(mydir)-1]
```

```python
if (len(resultjpg) != len(uniq) or len(resultcycle) != len(uniq)):
    print "{} ERROR: Length mismatch".format(ThisProg)
    sys.exit(1)
print "{} INFO: Reading {}".format(ThisProg,hscthin)
hscthout = hscthin+DEFINDCSVFILE
print "{} INFO: Writing {}".format(ThisProg,hscthout)
rownum=-1
with open(hscthin,'rb') as csvin, open(hscthout,'wb') as csvout:
    hscthopenin = csv.reader(csvin, delimiter=',', skipinitialspace=True)
    hscthopenout= csv.writer(csvout)
    iout=0
    for row in hscthopenin:
        rownum+=1
        if rownum > 2:
            cthcycle = int(row[1])
            numhits=0
            for m in resultcycle:
                numhits += m.count(cthcycle)
            if numhits != len(resultcycle):
                continue
            else:
                jpgindex = []
                famindex = 0
                for m in resultcycle:
                    jpgindex.append(m.index(cthcycle))
                for m in resultjpg:
                    if args.tool == 'Slycat':
                        row.insert(0,args.server+':'+myfulldir+'/'+m[jpgindex[
famindex]])
                    elif args.tool == 'Chorus':
                        row.insert(0,myreldir+'/'+m[jpgindex[famindex]])
                    else:
                        row.insert(0,args.server+':'+myfulldir+'/'+m[jpgindex[
famindex]])
                for m in range(0,args.numUser):
                    row.insert(0,0)
                for m in myparamsin[1]:
                    row.insert(0,m)
                iout += 1
                row.insert(0,iout)
                row.insert(0,mydir)
            #print "{} INFO: Row: {}, Cycle: {}".format(ThisProg,rownum,cthcycle)
        elif rownum == 1:
            row1 = list(row)
            continue
        elif rownum == 2:
            for m in range(0,len(row)):
                row[m] = row1[m]+'; '+row[m]
            for m in uniq:
                row.insert(0,m)
            for m in range(0,args.numUser):
                row.insert(0,'User'+str(m))
            for m in myparamsin[0]:
                row.insert(0,m)
                row.insert(0,'OutputNum')
                row.insert(0,'Identifier')
                if onlyprinti < onlyprintmax:
                    print "{} INFO: There will be {} columns in resultant CSV file".format(ThisProg,len(row))
            else:
                continue
            hscthopenout.writerow(row)
        csvin.close()
        csvout.close()



# This function is mapped to --hscth
def hscth():
    #Walk the file system
    print "{} INFO: I will recursively walk this directory ({}) and find CTH hscth, CTH Spymaster {} images, and Dakota params.in files and correlate them.".format(ThisProg,ThisDir,args.formatImage)
    p = []
    d = []
    f = []
    i=0
    for (dirpath, dirname, filename) in os.walk(ThisDir):
        i+=1
        print "{} INFO: Walking directory number {}\r".format(ThisProg,i),
        #print "dirpath = {}".format(dirpath)
        #print "dirname = {}".format(dirname)
        #print "filename = {}".format(filename)
        p.append(dirpath)
        d.append(dirname)
        f.append(filename)
    print "                                                                      "

    #Gather statistical information
    TotalDirs = len(p)
    TotalFiles = 0
    for i in range(0,len(f)):
        TotalFiles += len(f[i])
    print "{} INFO: I found {} directories.".format(ThisProg,TotalDirs)
    print "{} INFO: I found {} files.".format(ThisProg,TotalFiles)

    #Find hscth files
    if args.formatImage == "JPEG":
        regexjpg=re.compile("^.*\d{"+str(CTHCycleDigits)+"}\.(jpg|JPG|jpeg|JPEG)$")
    elif args.formatImage == "PNG":
        regexjpg=re.compile("^.*\d{"+str(CTHCycleDigits)+"}\.(png|PNG)$")
    for i in range(0,len(f)):
        resultjpg   = []
        resultcycle = []
        uniq        = []
        if "hscth" in f[i]:
            print "{} INFO: I found hscth within {}.".format(ThisProg,p[i])
            jpg = regexList(f[i],regexjpg)
            print "{} INFO: I found {} {} files.".format(ThisProg,len(jpg),args.
    formatImage)
            (resultjpg,resultcycle,uniq) = spyParse(jpg)
            hscthmod(resultjpg,resultcycle,uniq,p[i]+'/hscth')



#THIS FINDS INDIVIDUAL CSV FILES AND COMBINES THEM INTO A SINGLE ONE AND A PIECEMEAL ONE
# This was written this way for speed... there is likely a 100% Python way of achieving
    the same
# results without incurring a speed hit, but I couldn't find it quickly...
def csvcombine():
    script_one = '''\
#This defines the name of the combined CSV file
NEWFILE=hscth.combined.csv

#This defines the name of the CSV files to search for
INFILE=hscth.individual.csv

#This defines how many CSV files to put into a piecemeal CSV file
NUMCSV=10

echo "INFO: Creating single, combined CSV file ${NEWFILE}"
HEADER="$( find . -name $INFILE -print -quit )"
awk 'NR==1' $HEADER > $NEWFILE
find . -type f -name $INFILE -print0 | xargs -0 -I file awk 'NR>1' file >>$NEWFILE

NEWFILE=${NEWFILE%.csv}
inmcsv=$NUMCSV
((inmcsv++))
jnmcsv=0
find . -type f -name $INFILE | sort -V |
while read file ; do
    ((inmcsv++))
    fileout="${NEWFILE}${jnmcsv}.csv"
    if test $inmcsv -gt $NUMCSV ; then
        inmcsv=1
        ((jnmcsv++))
        fileout="${NEWFILE}${jnmcsv}.csv"
        echo "INFO: Creating piecemeal CSV file (${fileout})"
        awk 'NR==1' $HEADER > $fileout
    fi
    awk 'NR>2' "${file}" >> $fileout
done

echo "INFO: Finished after $SECONDS seconds"
'''
    SCRIPTNAME='combinecsv.sh'
    if args.writeScripts:
        print "{} INFO: Will create CSV concatenation script ({}) and not execute.".
    format(ThisProg,SCRIPTNAME)
        with open(SCRIPTNAME,'wb') as MYSCRIPT:
            MYSCRIPT.write(script_one)
            MYSCRIPT.close()
        return
    if os.path.isfile(SCRIPTNAME):
        print "{} INFO: Found CSV concatenation script ({}), will use that.".format(
    ThisProg,SCRIPTNAME)
        subprocess.call(['/bin/bash',SCRIPTNAME])
    elif os.path.isfile('/bin/bash'):
        print "{} INFO: Did not find CSV concatenation script ({}); will use built-in
    script.".format(ThisProg,SCRIPTNAME)
        with tempfile.NamedTemporaryFile() as scriptfile:
            scriptfile.write(script_one)
            scriptfile.flush()
            subprocess.call(['/bin/bash', scriptfile.name])
            scriptfile.close()
    else:
        print "{} WARNING: BASH and CSV concatenation script ({}) not found so aborting
    the CSV concatenation.".format(ThisProg,SCRIPTNAME)



#THIS READS IN COMBINED CSV FILE AND WRITES OUT ONLY THE ROWS WHERE "User#" VARIABLES ARE
     NONZERO
def csvfilter():
    regexuser=re.compile("^User\d+$")
    if os.path.isfile(csvcombined):
        with open(csvcombined,'rb') as csvin, open(csvfiltered,'wb') as csvout:
            csvopenin = csv.reader(csvin, delimiter=',', skipinitialspace=True)
            csvopenout= csv.writer(csvout)
            irow=0
            users=[]
            usersindex=[]
            for row in csvopenin:
                irow += 1
                if irow == 1:
                    users = regexList(row,regexuser)
                    for user in users:
                        usersindex.append(row.index(user))
                    csvopenout.writerow(row)
                else:
                    userstmp=[]
                    for m in usersindex:
                        userstmp.append(int(row[m]))
                    if any(v != 0 for v in userstmp):
                        csvopenout.writerow(row)
            csvin.close()
            csvout.close()
    else:
        print "{} WARNING: Combined CSV file ({}) not found so aborting the CSV filtering
    .".format(ThisProg,csvcombined)



#THIS BATCH PROCESSES A LOT OF IMAGES FOR LOADING AND SAVING EFFICIENCY
# This was written this way for speed... there is likely a 100% Python way of achieving
    the same
```

```
# results without incurring a speed hit, but I couldn't find it quickly...
def picfix():
    script_one = '''\
#This script will crop and resize Spymaster output from
#CTH example with ImageMagick.

export MAGICK_THREAD_LIMIT=8

#GOOD PERFORMANCE BY SETTING NUMBER OF THREADS EQUAL TO 2X AVAILABLE
if test -f /proc/cpuinfo ; then
    NumThreads=$( grep processor /proc/cpuinfo | wc -l )
    NumThreads=${NumThreads:-8}
    NumThreads=$((NumThreads * 2))
else
    NumThreads=8
fi
if test $NumThreads -gt 64 ; then
    NumThreads=64
fi
echo "INFO: Will create $NumThreads threads at a time"

convertme ()
{
    if test ! -f "${1%.jpg}".png ; then
        convert "${1}" -trim +repage -resize 360000@ -gravity center "${1%.jpg}".png
    fi
}
export -f convertme

main ()
{
    declare -i i
    declare -i j
    i=0
    j=0
    find . -type f -name "*.jpg" |
    while read file ; do
        if test $i -ge $NumThreads ; then
            echo -ne "No. of files processed (displayed every $NumThreads files): ${j}"\r
            wait
            i=0
        fi
        convertme "${file}" &
        ((i++))
        ((j++))
        #echo "Thread number: ${i}"
        #echo -ne "Number of files processed: ${j}"\r
    done
    wait
    echo
    echo "INFO: Finished with the following timing information:"
}
export -f main

```

```
time main

exit 0
'''
    SCRIPTNAME='fixpics.sh'
    if args.writeScripts:
        print "{} INFO: Will create image formatting script ({}) and not execute.".format
    (ThisProg,SCRIPTNAME)
        with open(SCRIPTNAME,'wb') as MYSCRIPT:
            MYSCRIPT.write(script_one)
            MYSCRIPT.close()
        return
    if os.path.isfile(SCRIPTNAME) and os.path.isfile('/bin/bash'):
        print "{} INFO: Found image formatting script ({}), will use that.".format(
    ThisProg,SCRIPTNAME)
        subprocess.call(['/bin/bash',SCRIPTNAME])
    elif os.path.isfile('/bin/bash'):
        print "{} INFO: Did not find image formatting script ({}); will use built-in
    script.".format(ThisProg,SCRIPTNAME)
        with tempfile.NamedTemporaryFile() as scriptfile:
            scriptfile.write(script_one)
            scriptfile.flush()
            subprocess.call(['/bin/bash', scriptfile.name])
            scriptfile.close()
    else:
        print "{} WARNING: BASH or image formatting script ({}) not found so aborting the
     image formatting.".format(ThisProg,SCRIPTNAME)



### DO WORK ###############################
# Create master CSV file from HSCTH and corresponding Spymaster JPEG files
if args.fixpics:
    picfix()
if args.hscth:
    hscth()
if args.combinecsv:
    csvcombine()
if args.filtercsv:
    csvfilter()



### EXIT ##################################
if len(sys.argv) == 1:
    print "{} HELP: No command line options were given; please pass either \"-h\" or \"--
    help\" to view my help page.".format(ThisProg)
    sys.exit(1)

print "{} INFO: These operations required {} minutes to complete.".format(ThisProg,(time.
    time()-start_time)/60)

sys.exit(0)
```

Script to create CSV file from `hscth` and Spymaster image output

Appendix F

Example Extending Chorus with Scripts

Tecplot provided some scripts to show how to extend Chorus to add post-processing and convenience features. These scripts are given below.

```xml
<?xml version="1.0"?>
<!DOCTYPE AuxFileActions>
<AuxFileActions>
  <action enabled="true">
    <name>Open With Chrome</name>
    <exec>"C:/Python27/python.exe" "C:\Program Files\Tecplot\Tecplot Chorus DE 2013 Beta\
    interop\actions\OpenWithChrome.py" "%INSTRUCTION_FILE%"</exec>
  </action>
  <action enabled="true">
    <name>Open Containing Folder</name>
    <exec>"C:/Python27/python.exe" "C:\Program Files\Tecplot\Tecplot Chorus DE 2013 Beta\
    interop\actions\OpenContainingFolder.py" "%INSTRUCTION_FILE%"</exec>
  </action>
  <action enabled="false">
    <name>Extract Data with Tecplot 360</name>
    <exec>"/path/to/python" "/path/to/ExtractDataWithTecplot360.py" "%INSTRUCTION_FILE%"<
    /exec>
  </action>
  <action enabled="false">
    <name>Open Data with Paraview</name>
    <exec>"/path/to/python" "/path/to/OpenWithParaview.py" "%INSTRUCTION_FILE%"</exec>
  </action>
</AuxFileActions>
```

XML file that registers Python functions

```python
# -*- coding: utf-8 -*-

import sys
import os
from JobInfo import *
import subprocess

instructionFile = sys.argv[1]
```

```python
assert(os.path.exists(instructionFile))

jobInfo = JobInfo()
jobInfo.fromXMLFile(instructionFile)

os.remove(instructionFile)

for caseID, dataFiles in jobInfo.dataFiles().iteritems():
    if len(dataFiles) > 0:
        fileName = os.path.abspath(dataFiles[0])

        # Windows only and only works for a single path
        instruction = r'explorer /select, "%s"' %(fileName)
        subprocess.Popen(instruction)
        break
```

Python function to open the folder of a selected point

```python
# -*- coding: utf-8 -*-

import sys
import os
from JobInfo import *
import subprocess

instructionFile = sys.argv[1]
assert(os.path.exists(instructionFile))

jobInfo = JobInfo()
jobInfo.fromXMLFile(instructionFile)

os.remove(instructionFile)

for caseID, dataFiles in jobInfo.dataFiles().iteritems():
    if len(dataFiles) > 0:
        fileName = os.path.abspath(dataFiles[0])

        # Windows only and only works for a single path
        instruction = r'C:\Users\scottf\AppData\Local\Google\Chrome\Application\chrome.
    exe "%s"' %(fileName)
        subprocess.Popen(instruction)
        break
```

Python function to open some data within Chrome

Appendix G

Obtaining Documentation and Scripts

This report, created with $\LaTeX\,2_\varepsilon$, and the source code of all scripts it contains is kept within a Git repository on Lynx. The command to download a copy of this repository is given below.

```
$ git clone lynx:/projects/hpc-scale/GITREPOS/EnsembleAnalysis.git
Cloning into 'EnsembleAnalysis'...
amagela@lynx's password:
remote: Counting objects: 110, done.
remote: Compressing objects: 100% (108/108), done.
remote: Total 110 (delta 17), reused 0 (delta 0)
Receiving objects: 100% (110/110), 7.65 MiB | 1.13 MiB/s, done.
Resolving deltas: 100% (17/17), done.
Checking connectivity... done.
```

Command and its typical output to obtain a copy of the repository

DISTRIBUTION:

1	MS 0807	Anthony M. Agelastos, 9326
1	MS 0807	Joel O. Stevenson, 9326
1	MS 0840	Stephen W. Attaway, 1555
1	MS 0840	John P. Korbin, 1555
1	MS 0840	David J. Peterson, 1555
1	MS 0899	Technical Library, 9536 (electronic copy)

Sandia National Laboratories